TUGAS AKHIR - KI141502

Implementasi Kecerdasan Buatan Pada Permainan "Phantom Crown" Menggunakan Hierarchical Finite State Machine dan Decision Tree

Dimas Rahman Oetomo
NRP 5113100163

Dosen Pembimbing I
Imam Kuswardayan, S.Kom, M.T.

Dosen Pembimbing II
Dr.Eng. Nanik Suciati, S.Kom, M.Kom

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

[Halaman ini sengaja dikosongkan]

TUGAS AKHIR - KI141502

Implementasi Kecerdasan Buatan Pada Permainan "Phantom Crown" Menggunakan Hierarchical Finite State Machine dan Decision Tree

Dimas Rahman Oetomo
NRP 5113100163

Dosen Pembimbing I
Imam Kuswardayan, S.Kom, M.T.

Dosen Pembimbing II
Dr.Eng. Nanik Suciati, S.Kom, M.Kom

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESES – KI141502

Artificial Intelligence Implementation For "Phantom Crown" Game Using Hierarchical Finite State Machine and Decision Tree

Dimas Rahman Oetomo
NRP 5113100163

Supervisor I
Imam Kuswardayan, S.Kom, M.T.

Supervisor II
Dr.Eng. Nanik Suciati, S.Kom, M.Kom.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology and Communication
Institut Teknologi Sepuluh Nopember
SURABAYA 2018

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

Implementasi Kecerdasan Buatan Pada Permainan "Phantom Crown" Menggunakan Hierarchical Finite State Machine dan Decision Tree

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer pada
Rumpun Mata Kuliah Interaksi, Grafika, dan Seni
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh :
Dimas Rahman Oetomo
NRP : 5113 100 163

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Imam Kuswardayan, S.Kom., M.T. .....
NIP: 19761215 200312 1 001 (pembimbing 1)

Dr.Eng. Nanik Suciati, S.Kom., M.Kom. .....
NIP: 19710428 199412 2 001 (pembimbing 2)

KEMENTERIAN RISET, TEKNOLOGI, DAN PENDIDIKAN TINGGI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
FAKULTAS TEKNOLOGI INFORMASI DAN KOMUNIKASI
DEPARTEMEN TEKNIK INFORMATIKA

SURABAYA
JANUARI 2018

[Halaman ini sengaja dikosongkan]

Implementasi Kecerdasan Buatan Pada Permainan "Phantom Crown" Menggunakan Hierarchical Finite State Machine dan Decision Tree

Nama Mahasiswa	**: Dimas Rahman Oetomo**
NRP	**: 5113100163**
Jurusan	**: Departemen Informatika FTIK-ITS**
Dosen Pembimbing 1	**: Imam Kuswardayan, S.Kom., M.T.**
Dosen Pembimbing 2	**: Dr.Eng. Nanik Suciati, S.Kom, M.Kom.**

ABSTRAK

Phantom Crown merupakan permainan di mana dua pemain saling bertarung. Agar Phantom Crown lebih menyenangkan dibuatlah lawan bermain yang memiliki kecerdasan buatan. Implementasi kecerdasan buatan dilakukan menggunakan Decision Tree (DT). DT dipilih karena berguna untuk mengeksplorasi data dan menemukan hubungan tersembunyi antara sejumlah calon variabel input dengan sebuah variabel target.

Untuk menunjang kecerdasan buatan yang dibuat, diperlukan alur bermain yang dibuat menggunakan Hierarchical Finite State Machine (HFSM). HFSM digunakan karena memudahkan programmer dalam mengimplementasikan alur yang sudah didesain dan membuat sistem tidak terlalu kompleks.

Eksperimen yang dilakukan kepada 6 orang menghasilkan persentase kepuasan pengguna sebesar 66%. Eksperimen tingkat kecerdasan buatan dilakukan dengan menandingkan musuh melawan musuh. Eksperimen menghasilkan 86% probabilitas menang untuk musuh dengan tingkat kecerdasan lebih tinggi..

Kata kunci: Decision Tree, Desain Permainan, Hierarchical Finite State Machine, Kecerdasan Buatan.

[Halaman ini sengaja dikosongkan]

Artificial Intelligence Implementation For "Phantom Crown" Game Using Hierarchical Finite State Machine and Decision Tree

Student's Name	**: Dimas Rahman Oetomo**
Student's ID	**: 5113100163**
Department	**: Departement of Informatics FTIK-ITS**
First Advisor	**: Imam Kuswardayan, S.Kom., M.T.**
Second Advisor	**: Dr.Eng. Nanik Suciati, S.Kom, M.Kom.**

ABSTRACT

Phantom Crown is a game where two players fight each other. In order to make Phantom Crown more fun, the Artificial Intelligence is applied to the enemy. The Decision Tree(DT) is used to choose the movement that will be performed by the enemy. DT is used because it is useful for data exploration and finding hidden relation between variables.

DT only is not enough to make good enemy, enemy need good state flow. The Hierarchical Finite State Machine (HFSM) is used because it allows programmer to execute the design flow easily and make system less complex.

Experiment of user satisfaction is conducted by six people, the experiment generate 66% of user satisfaction. Experiment of artificial intelligence is conducted by allowing enemy fight each other, the experiment generate 86% of win probality for enemy with better artificial intelligence.

Keyword: Artificial Intelligence, Decision Tree, Game Design, Hierarchical Finite State Machine.

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji dan syukur bagi Allah SWT, yang telah melimpahkan rahmat, dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul

Implementasi Enemy AI Pada Game "Phantom Crown" dengan Menggunakan Metode Hierarchical Finite State Machine dan Decision Tree

Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang berharga bagi penulis. Dengan pengerjaan Tugas Akhir, penulis dapat memperdalam, meningkatkan, serta menerapkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Teknik Informatika ITS.

Terselesaikannya Tugas Akhir ini tidak lepas dari bantuan dan dukungan dari berbagai pihak. Dalam kesempatan ini penulis mengucapkan rasa syukur dan terima kasih kepada:

1. Allah SWT, karena atas izin-Nya lah penulis dapat menyelesaikan Tugas Akhir dengan baik.
2. Kedua orang tua, Ir. Hj. Muhammad Rifai serta Dra. Hj. Maria Ulfah dan ketiga saudara Choirunisa Rifa Yuliani, Dara Ninggar Hanifa, serta Ghaisan Aulia Putra. terima kasih atas doa dan bantuan moral dan material selama penulis belajar di Teknik Informatika ITS.
3. Bapak Dr. Darlis Herumurti, S.Kom., M.Kom., selaku ketua jurusan Teknik Informatika ITS
4. Bapak Radityo Anggoro, S.Kom., M.Sc. selaku Koordinator Tugas Akhir di Teknik Informatika ITS.
5. Bapak Radityo Anggoro, S.Kom., M.Sc. selaku koordinator TA.
6. Bapak Imam Kuswardayan, S.Kom., M.T.selaku pembimbing I Tugas Akhir yang telah memberikan banyak waktu untuk

berdiskusi dan memberi semangat dan motivasi kepada penulis untuk menyelesaikan Tugas Akhir.

7. Ibu Dr.Eng. Nanik Suciati, S.Kom, M.Kom selaku pembimbing II Tugas Akhir yang telah memberikan banyak yang telah memberikan bimbingan dan dukungan selama penulis menyelesaikan Tugas Akhir.
8. Bapak dan Ibu Dosen di Jurusan Teknik Informatika yang telah memberikan ilmu selama penulis kuliah di Teknik Informatika
9. Seluruh Staf dan karyawan Teknik Informatika yang telah memberikan bantuan selama penulis kuliah di Teknik Informatika.
10. Rekan-rekan dan pengelola Laboratorium Interaksi, Grafik, dan Seni yang telah memberikan fasilitas dan kesempatan melakukan riset atas Tugas Akhir yang dikerjakan penulis.
11. Rekan-rekan di UKM Kendo yang memberikan semangat dan motivasi.
12. Rekan-rekan dan sahabat-sahabat penulis angkatan 2013 yang memberikan dorongan motivasi dan bantuan kepada penulis.
13. Pihak-pihak lain yang tidak sengaja terlewat dan tidak dapat penulis sebutkan satu per satu.

Penulis memohon maaf apabila terdapat kekurangan dalam penulisan Tugas Akhir ini. Kritik dan saran penulis harapkan untuk perbaikan dan pembelajaran di kemudian hari. Semoga Tugas Akhir ini dapat memberikan Manfaat yang sebesar besarnya.

Surabaya, Januari 2018

Dimas Rahman Oetomo

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

DAFTAR KODE SUMBER

BAB I
PENDAHULUAN

Pada bab ini dibahas mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika laporan tugas akhir. Diharapkan dari penjelasan dalam bab ini gambaran tugas akhir secara umum dapat dipahami.

1.1. Latar Belakang

Phantom Crown merupakan *fighting game* yang akan dirancang pada tugas akhir ini. *Fighting game* adalah s*ubgenre video game* dari *genre Action Game* dimana pemain mengkontrol sebuah karakter yang bertarung melawan musuh dipertarungan jarak dekat [1]. Pertarungan di dalam *fighting game* biasanya terdiri dari beberapa ronde. Agar permainan menjadi lebih menyenangkan musuh dalam permainan dapat dikendalikan oleh kecerdasan buatan.

Untuk membuat kecerdasan buatan yang dapat berkompetisi dengan manusia diperlukan alur pemodelan kebiasaan manusia ketika bermain permainan phantom crown. Untuk mengatasi itu penulis menggunakan HFSM seperti yang telah dilakukan oleh Peter M. Kielar, Oliver Handel, Daniel H. Biedermann, dan Andre Borrman pada penelitian mereka yaitu pemodelan kebiasaan pejalan kaki [2]. Pemodelan alur manusia belum cukup untuk membuat musuh permainan yang kompeten. Oleh karena itu penulis memilih DT sebagai kecerdasan buatan untuk memberikan prediksi pergerakan selanjutnya. Penggunaan DT untuk memprediksi telah dilakukan oleh Gerasimos Spanakis dkk dalam riset mereka [3].

Tujuan dari pengerjaan tugas akhir ini adalah mampu menghasilkan lawan bermain untuk *fighting game* yang memiliki kecerdasan buatan. Lawan bermain yang dihasilkan diharapkan untuk menjadi dasar untuk pengembangan selajutnya. Selain itu, tugas akhir ini diharapkan mampu memberikan solusi terhadap penerapan HFSM dan DT.

1.2. Rumusan Permasalahan

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana merancang aturan dan tingkat kesulitan dalam permainan Phantom Crown?
2. Bagaimana bentuk pemodelan ketika bermain "Phantom Crown" menggunakan *Hierarchical Finite State Machine*?
3. Bagaimana bentuk implementasi *Hierarchical Finite State Machine* di dalam permainan Phantom Crown?
4. Bagaimana membuat kecerdasan buatan dengan *Decision Tree* sehingga pemain mendapat kesulitan?

1.3. Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Pembuatan *tree* akan dibantu dengan *library* Linq.
2. Aplikasi akan dijalankan pada IDE Unity.
3. Bahasa pemograman yang digunakan adalah C# yang telah dimodifikasi untuk Unity.

1.4. Tujuan

Tujuan dari pembuatan tugas akhir ini adalah merancang kecerdasan buatan untuk permainan yang dapat dikembangkan untuk permainan yang lebih kompleks.

1.5. Manfaat

Pengerjaan tugas akhir ini dilakukan dengan harapan dapat memberikan kontribusi sebagai dasar untuk para pengembang permainan ketika membuat kecerdasan buatan yang lebih kompleks untuk permainan dengan *genre action 2D.*

1.6. Metodologi

Pembuatan tugas akhir dilakukan menggunakan metodologi sebagai berikut:

1. Penyusunan proposal Tugas Akhir.

   Proposal tugas akhir berisi tentang deskripsi pendahuluan dari tugas akhir yang akan dibuat. Pendahuluan ini terdiri atas latar belakang, rumusan masalah tugas akhir, batasan masalah yang diangkat, tujuan, serta manfaat dari hasil pembuatan tugas akhir. Selain itu dijabarkan juga tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan tugas akhir. Sub bab ringkasan isi yang berisi ringkasan singkat tentang usulan tugas akhir yang akan dikerjakan, serta sub bab metodologi yang berisi penjelasan tentang tahap penyusunan tugas akhir dan jadwal pengerjaan tugas akhir.

2. Studi literatur

   Tahap studi literatur merupakan tahap pembelajaran dan pengumpulan informasi yang digunakan untuk mengimplementasikan tugas akhir. Tahap ini diawali dengan pengumpulan literatur, diskusi dengan sumber, eksplorasi teknologi dan pustaka, serta pemahaman dasar teori yang digunakan pada topik tugas akhir.

3. Perancangan perangkat lunak

   Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini mendefinisikan alur dari implementasi. Langkah-langkah yang dikerjakan juga didefinisikan pada tahap ini. Pada tahapan ini dibuat *prototype* sistem, yang merupakan rancangan dasar dari sistem yang akan dibuat. Kemudian dilakukan desain suatu sistem dan desain proses-proses yang ada.

4. Implementasi perangkat lunak
   Implementasi merupakan tahap membangun rancangan program yang telah dibuat. Pada tahapan ini merealisasikan rancangan yang terdapat pada tahapan sebelumnya, sehingga menjadi sebuah program yang sesuai dengan apa yang telah direncanakan.

5. Pengujian dan evaluasi
   Pada tahap ini dilakukan uji coba terhadap perangkat lunak yang telah dibuat untuk mengetahui kemampuan algoritma yang dipakai, mengamati kinerja sistem, serta mengidentifikasi kendala yang mungkin timbul pada aplikasi yang dibuat.

6. Penyusunan buku Tugas Akhir.
   Pada tahapan ini disusun buku yang memuat dokumentasi mengenai pembuatan serta hasil dari implementasi perangkat lunak yang telah dibuat.

1.7. Sistematika Penulisan Laporan Tugas Akhir

Buku tugas akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan tugas akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku tugas akhir terdiri atas beberapa bagian seperti berikut ini.

1. Bab I. Pendahuluan
   Bab ini berisi penjelasan mengenai latar belakang masalah, tujuan, dan manfaat dari pembuatan tugas akhir. Selain itu rumusan permasalahan, batasan masalah, dan sistematika penulisan juga merupakan bagian dari bab ini.
2. Bab II Tinjauan Pustaka
   Bab ini membahas dasar pembuatan dan beberapa teori penunjang yang berhubungan dengan pokok pembahasan yang mendasari pembuatan tugas akhir ini.

3. Bab III Perancangan Perangkat Lunak
   Bab ini membahas analisis dari sistem yang dibuat meliputi analisis permasalahan, deskripsi umum perangkat lunak, spesifikasi kebutuhan, dan identifikasi pengguna. Kemudian membahas rancangan dari sistem yang dibuat meliputi rancangan skenario kasus penggunaan, arsitektur, data, dan antarmuka.

4. Bab IV. Implementasi
   Bab ini merupakan pembangunan aplikasi dengan Unity 5 sesuai permasalahan dan batasan yang telah dijabarkan pada Bab I.

5. Bab V. Hasil Uji Coba dan Evaluasi
   Bab ini berisi penjelasan mengenai data hasil percobaan, pengukuran, dan pembahasan mengenai hasil percobaan yang telah dilakukan.

6. Bab VI. Kesimpulan dan Saran
   Bab ini berupa hasil penelitian yang menjawab permasalahan atau yang berupa konsep, program, dan karya rancangan. Selain itu, pada bab ini diberikan saran-saran yang berisi hal-hal yang masih dapat dikerjakan dengan lebih baik dan dapat dikembangkan lebih lanjut, atau berisi masalah-masalah yang dialami pada proses pengerjaan Tugas Akhir.

[Halaman ini sengaja dikosongkan]

BAB II
DASAR TEORI

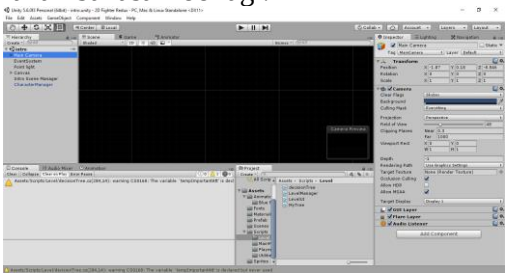
Bab ini berisi penjelasan teori-teori yang berkaitan dengan pembuatan kecerdasan buatan untuk lawan bermain dalam permainan Phantom Crown. Penjelasan ini bertujuan untuk memberikan dasar teori yang mendasari pengembangan perangkat lunak.

2.1. Unity3D

Unity3D merupakan sebuah *game engine* yang dikembangakan oleh Unity Technology [4]. Unity3D dapat membuat aplikasi untuk berbagai *platform* seperti Android, Ios, Windows, Playstation, Nintendo dan Xbox [5]. Unity3D ditujukan untuk membuat arsitektur bangunan, simulasi serta permainan berbasis 2D atau 3D.

Unity3D menggunakan sebuah konsep yang disebut *parenting*. Ini digunakan untuk membuat sebuah *game object* dapat menjadi anak dari *game object* yang lain. Tarik sebuah *game object* dan pindahkan tepat di atas tulisan *game object* yang akan dijadikan *parent* dalam *hierarchy*. *game object* yang terdapat dalam sebuah *game object* lainnya akan mengikuti perpindahan dan perputaran ketika *game object parent* mengalami perubahan posisi.

Karena untiy3D menggunakan konsep *parenting* sehingga diperlukan bahasa pemograman yang dapat menunjang pemrograman berorientasi objek. Bahasa pemrograman yang disediakan unity3D adalah JavaScript, C#, dan Boo. Namun mulai dari versi 5.0, unity3D tidak menyediakan bahasa Boo lagi.



Gambar 2.1 Tampilan dalam Program Unity3D 5

Gambar 2.1 menampilkan lembar kerja dari unity3D 5 versi gratis. Pada setiap *project* Unity3D terdapat sebuah *assets folder*. Isi dari *assets folder* ditampilkan dalam bentuk panel *project* dalam *editor* unity. *Assets folder* adalah tempat untuk menyimpan semua komponen dari permainan seperti *level scenes*, *scripts*, 3D *models*, tekstur, dan *file audio*.

Panel *hierarchy* menampung semua *game object* yang terdapat di *scene* yang sedang aktif. Beberapa dari *game object* tersebut berhubungan langsung ke *asset* seperti objek 3D. Objek yang terdapat pada *hierarchy* dapat di seleksi dan dihapus. Jika objek dihapus atau ditambahkan pada *scene*, maka objek tersebut juga akan hilang atau muncul pada *hierarchy*.

2.2. Clip Studio Paint

Clip Studio Paint (sebelumnya dipasarkan sebagai Manga Studio di Amerika Utara dan ComicStudio di Jepang) adalah sebuah perangkat lunak untuk macOS dan Microsoft Windows yang digunakan untuk pembuatan komik dan illustrasi. Clip Studio Paint dikembangkan oleh Celsys, perusahaan perangkat lunak grafis Jepang. Versi saat ini dijual sebagai "Clip Studio Paint Pro", dan "Clip Studio Paint EX" yang menambahkan dukungan untuk dokumen multihalaman dan fitur lainnya [6].

2.3. Bahasa C#

Bahasa C# yang dibaca C sharp merupakan bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bahasa pemrograman untuk .NET Framework [7].

Bentuk dari struktur bahasa C# adalah sebagai berikut:

- *Resource* atau *Library*
  Struktur pertama ini kita mendefinisikan *library* atau program apa saja yang ingin digunakan pada program yang akan dibuat.
- *Namespace*

Namespace yang merupakan struktur kedua ini adalah nama dari proyek yang dibuat.

- *Class*
  Struktur ketiga ini cetak biru untuk membuat objek. Kelas menentukan apa yang dimiliki sebuah objek (atribut) dan apa yang dapat dilakukan objek.
- Deklarasi *Method*
  Struktur keempat merupakan pendeklarasian *method* yang bertujuan untuk menjalankan perintah yang ada di dalam *method*.
- *Method* atau *Command*
  Struktur kelima adalah *Method* atau perintah adalah bagian – bagian kode yang akan dipanggil oleh program utama atau dari *method* lainnya untuk menjalankan fungsi yang spesifik.

```
using System;

namespace BelajarCSHARP
{
 class programHW
 {
  static void Main(string[] args)
  {
   Console.WriteLine("Hello World!");

          Console.Write("Press any key to continue . . . ");
             Console.ReadKey(true);
  }
 }
}
```

Gambar 2.2 Contoh Program C#

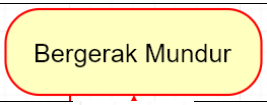
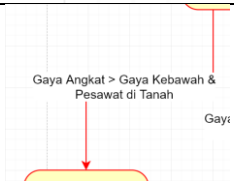
Pada contoh diatas System adalah *library* yang digunakan, selanjutnya struktur kedua adalah BelajarCSHARP sebagai nama dari proyek. BelajarCSHARP tersebut memiliki sebuah *class* bernama programHW yang berisikan *method* bernama main. *Method main* memiliki sebuah perintah atau *command* untuk menulis “Hello World!”.

2.4. Hierarchical Finite State Machine

Hierarchial Finite State Machine atau *StateCharts* merupakan hasil pengembangan dari *Finite State Machine* yang dibuat oleh Moore(1956) dan dipopulerkan oleh Harel(1987), HFSM merupakan metode yang efisien untuk memodelkan sistem yang reaktif dan merupakan standar di dunia *software engineering* untuk sistem yang reaktif.

Karena dikembangkan dari FSM, HFSM memiliki 3 prinsip yang sama dengan FSM yaitu *State*, *Event*, dan *Action.* Sistem dapat berganti ke *state* yang lain jika mendapatkan *input* yang telah ditentukan atau karena sebuah *event* yang berasal dari *device* eksternal atau komponen sistem itu sendiri. Kondisi perpindahan ini umumnya diikuti dengan sebuah *action*.

Tabel 2.1 Simbol Pada FSM

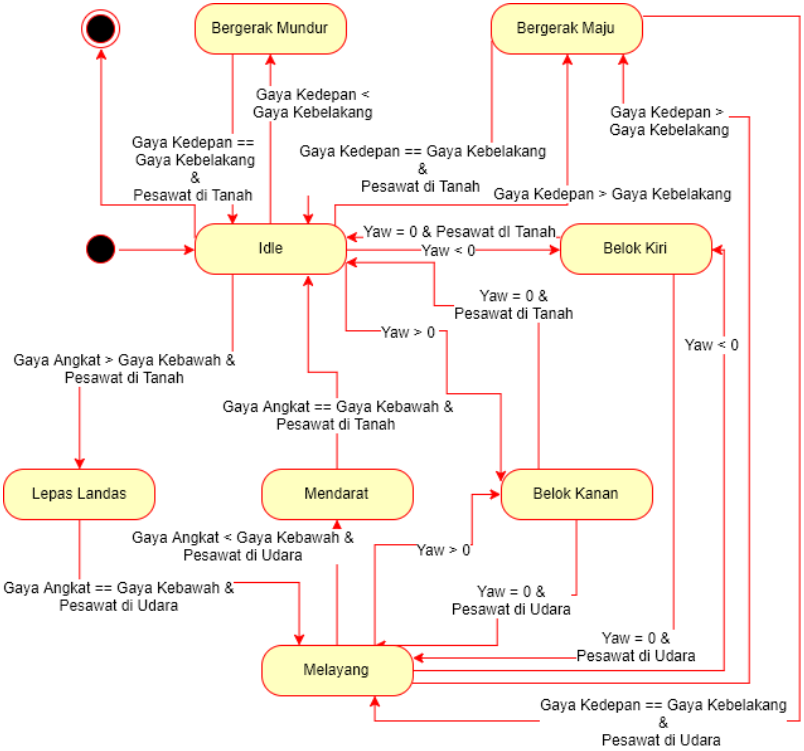
	*Start*
	*State*
	*End*
	*Transition*

Perbedaan FSM dan HFSM adalah *superstates*. *Superstates* merupakan kumpulan *state* yang saling terkoneksi melalui transisi. Dengan adanya *superstates* hanya akan melakukan satu transisi

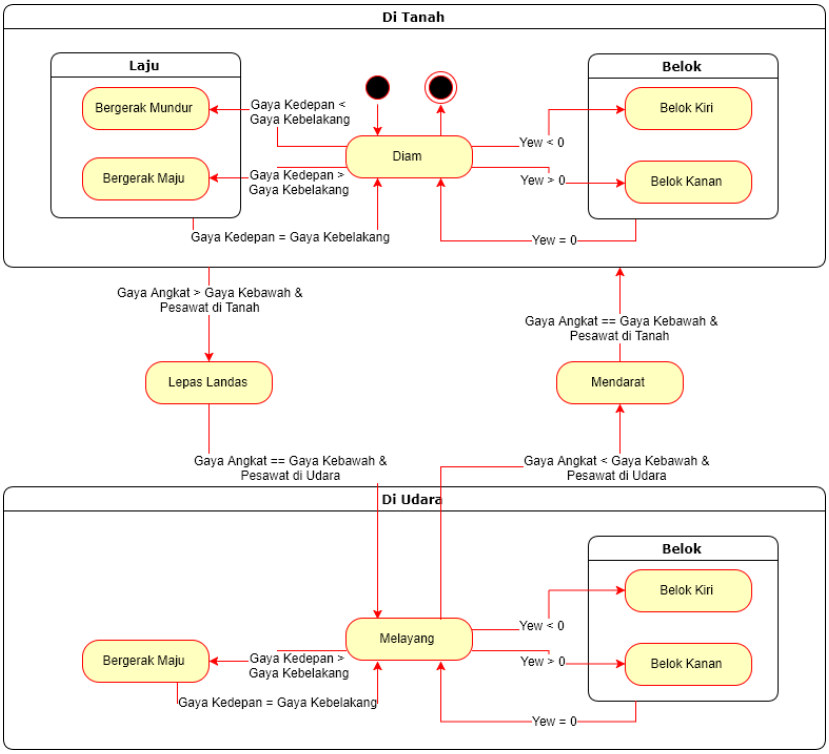
dengan *superstates* lainnya, sehingga dapat mengurangi redudansi transisi [8].

Pengelompokan *state* ke dalam *superstates* dilakukan berdasarkan *behavior* atau tingkah laku. Karena adanya pengelompokkan tingkah laku sehingga memudahkan klusterisasi tingkah laku yang berguna dalam prediksi dan analisa masalah.

Perbedaan bentuk FSM dan HFSM di dalam pergerakan pesawat dapat dilihat sebagai berikut:



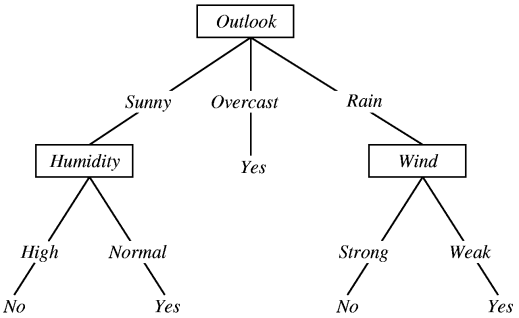
Gambar 2.3 Konsep FSM Dalam Pergerakan Pesawat



Gambar 2.4 Konsep HFSM Dalam Pergerakan Pesawat

2.5. Decision Tree

Decision Tree merupakan metode klarifikasi yang mudah diinterpretasikan. DT adalah model prediksi menggunakan struktur pohon atau struktur berhirarki. Setiap percabangan DT menyatakan kondisi yang harus dipenuhi dan setiap ujung pohon menyatakan kelas data. Contoh untuk DT dapat dilihat di Gambar 2.5 Model *Decision Tree* berikut ini.



Gambar 2.5 Model *Decision Tree*

Konsep dari DT adalah mengubah data menjadi pohon keputusan dan aturan-aturan keputusan. Manfaat utama dari penggunaan DT adalah kemampuannya untuk mem-*breakdown* proses pengambilan keputusan yang kompleks menjadi lebih simpel sehingga pengambil keputusan akan lebih menginterpretasikan solusi dari permasalahan [9].

DT juga berguna untuk mengeksplorasi data, menemukan hubungan tersembunyi antara sejumlah calon variabel input dengan sebuah variabel target. DT memadukan antara eksplorasi data dan pemodelan, sehingga sangat bagus sebagai langkah awal dalam proses pemodelan bahkan ketika dijadikan sebagai model akhir dari beberapa teknik lain..

Dalam beberapa aplikasi, akurasi dari sebuah klasifikasi atau prediksi adalah satu-satunya hal yang ditonjolkan, misalnya sebuah perusahaan directmail membuat sebuah model yang akurat untuk memprediksi anggota mana yang berpotensi untuk merespon permintaan, tanpa memperhatikan bagaimana atau mengapa model tersebut bekerja.

2.5.1. Algoritma Iterative Dichotomiser Three(ID3)

Iterative Dichotomiser 3 (ID3) merupakan sebuah metode yang digunakan untuk membuat DT yang telah dikembangkan oleh J. Ross Quinlan sejak tahun 1986. Algoritma pada metode ini menggunakan konsep dari *entropy* informasi. Algoritma ini

melakukan pencarian secara rakus(*greedy*) pada semua kemungkinan pohon keputusan.

Secara ringkas, langkah kerja algoritma ID3 dapat digambarkan sebagai berikut:

1. Hitung *information gain* dari setiap atribut dan pilih nilai terbesar.
2. Buat simpul yang berisi atribut tersebut.
3. Ulangi proses perhitungan *information gain* yang akan terus dilaksanakan sampai semua cabang memiliki *leaf*. Jika atribut ada dalam percabangan yang sama maka tidak diikutkan lagi dalam perhitungan *information gain*.

2.5.2. Information Gain

Information gain adalah salah satu *atribute selection measure* yang digunakan untuk memilih atribut tes tiap *node* pada *tree*. Atribut dengan *information gain* tertinggi dipilih sebagai atribut tes dari suatu node [10].

Rumus dari *information gain* adalah sebagai berikut:

$$G(S,A) = H(S) - \sum_{V \in Values(A)} \frac{|S_v|}{|S|} H(S_v) \qquad (2.1)$$

Keterangan:

$\boldsymbol{V}$: Merupakan bagian nilai dari *A*.

$\boldsymbol{S}$: Contoh *node* yang sedang dipelajari.

$\boldsymbol{S_v}$: jumlah nilai dari *V*.

$\boldsymbol{H}$: Merupakan *entropy*.

Entropy yang merupakan bagian dari *information gain* adalah formula untuk menghitung homogenitas dari sebuah *sample*/contoh.

Prinsip dari *entropy* adalah sebagai berikut:

$$H(S) = -p_+ log_2 p_+ - p_- log_2 p_- \quad (2.2)$$

Keterangan:

$\boldsymbol{H}$: Merupakan *entropy* .

$\mathbf{S}$: Contoh *node* yang sedang dipelajari.

$\boldsymbol{p_+}$: jumlah nilai yang positif.

$\boldsymbol{p_-}$: jumlah nilai yang negatif.

[Halaman ini sengaja dikosongkan]

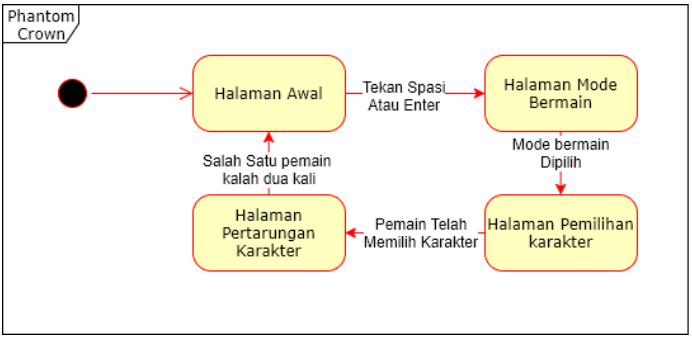
BAB III
PERANCANGAN PERANGKAT LUNAK

Pada bab ini dijelaskan mengenai rancangan kecerdasan buatan yang akan dibuat. Perancangan desain alur kecerdasan buatan menggunakan HFSM. Perancangan kecerdasan buatan meliputi data untuk DT, serta bentuk dari *tree*. Hasil dari perancangan HFSM dan DT digunakan sebagai pembelajaran maupun pengujian sehingga tujuan tugas akhir ini bisa tercapai.

3.1. Perancangan Permainan

3.1.1. Deskripsi Umum perangkat Lunak

Tugas akhir dikembangkan sebuah permainan bernama Phantom Crown memiliki *genre fighting game*. Permainan dimana tempat kecerdasan buatan berjalan dibuat dengan HFSM. *state–state* yang dibuat akan dikelompokkan ke dalam *state* yang lebih besar yang dapat disebut *superstate*. *Superstate* yang terbesar akan dibagi menjadi 4 yaitu *superstate* halaman awal, *superstate* halaman mode bermain, *superstate* halaman pemilihan karakter, dan *superstate* halaman pertarungan karakter seperti yang terlihat pada Gambar 3.1 Gambaran Besar H.



Gambar 3.1 Gambaran Besar HFSM Phantom Crown

Permainan dimulai dari halaman awal yang berisi judul permainan. Jika pemain menekan spasi atau enter pada *keyboard*, maka akan diarahkan ke halaman mode bermain. Di halaman mode bermain terdapat beberapa mode manusia melawan manusia, manusia melawan kecerdasan buatan dan kecerdasan buatan melawan kecerdasan buatan. Setelah memilih mode bermain, maka akan diarahkan ke halaman pemilihan karakter. Di halaman pemilihan karakter pemain akan melakukan pemilihan karakter. Setelah memilih pemain memilih karakter, maka akan diarahkan ke halaman pertarungan karakter. Di halaman pertarungan karakter, pemain akan saling bertarung hingga salah satu pemain kalah dua ronde. Di *state* ini tugas akhir yaitu kecerdasan buatan untuk karakter dibuat. Ketika salah satu pemain kalah dua ronde, maka akan diarahkan kembali ke halaman awal.

3.1.2. Spesifikasi Kebutuhan Fungsional

Berdasarkan deskripsi umum perangkat lunak, maka dapat disimpulkan bahwa kebutuhan fungsional pada perangkat lunak ini adalah:

1. Sistem harus dapat menampilkan permainan.
2. Sistem memungkinkan pengguna untuk memilih mode permainan.
3. Sistem harus memungkinkan pemain untuk mengerakkan karakter yang dipilih.
4. Sistem harus dapat menentukan pemenang dari permainan.

3.1.3. Spesifikasi Kebutuhan Non-Fungsional

Terdapat beberapa kebutuhan non-fungsional yang perlu dipenuhi sehingga program dapat berjalan dengan baik. Berikut kebutuhan non-fungsional:

1. Sistem berjalan secara *offline*.
2. Sistem berjalan pada *platform* Unity 5.
3. Sistem memerlukan Grafis 2D berbentuk jpg atau png.
4. Sistem harus memiliki *FrameRate* minimum 24.

3.1.4. Perancangan Aturan Permainan

Aturan permainan adalah serangkaian aturan yang harus diikuti pemain untuk memperoleh kemenangan. Aturan permainan yang dirancang adalah sebagai berikut.

- Kemenangan diperoleh kepada pemain dengan kesehatan terbanyak.
- Pemain dapat mengurangi kesehatan musuh dengan pukulan dan tendangan.
- Waktu bermain setiap ronde adalah 30 detik.
- Pemain dapat bergerak ke kiri dan ke kanan.
- Jika pemain terkena serangan ketika bergerak menjauh dari musuh, pemain akan menangkis serangan sehingga kesehatan pemain tidak berkurang.

3.2. Perancangan Alur Untuk Kecerdasan Buatan

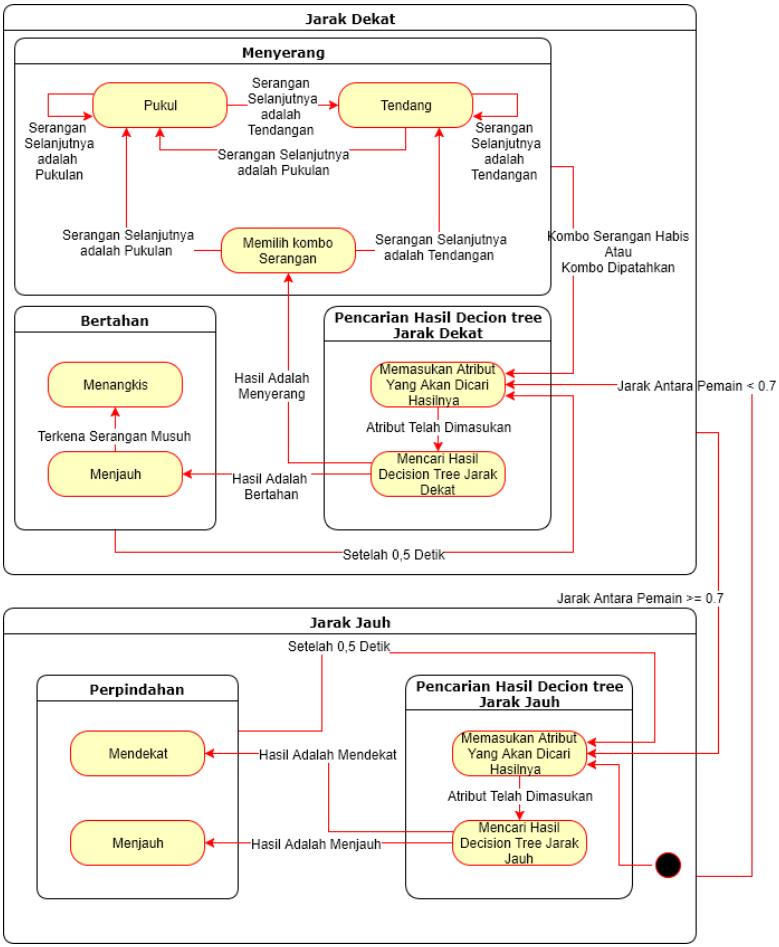
Pada bagian ini menjelaskan alur untuk pemodelan kecerdasan buatan untuk kecerdasan buatan. Perancangan pemodelan alur akan menggunakan *Hierarchical Finite State* Machine(HFSM).

Dalam alur HFSM yang telah dibuat, alur memiliki 2 *superstate* terbesar yaitu *superstate* jarak dekat dan *superstate* jarak jauh. *Superstate* jarak dekat memiliki fungsi untuk menggerakkan karakter ketika karakter berada di dalam jarak serang. *Superstate* jarak jauh memiliki fungsi untuk menggerakkan karakter ketika karakter berada diluar jarak serang. *Superstate* jarak dekat dan *superstate* jarak jauh memiliki *superstate* pencarian hasil decision tree yang berguna untuk memilih pergerakan yang akan dilakukan oleh karakter.

Superstate jarak dekat memiliki 2 *superstate* yang digunakan untuk menggerakkan karakter yaitu *superstate* menyerang dan *superstate* bertahan. *Superstate* menyerang berisi *state* untuk mengurangi kesehatan musuh. *Superstate* bertahan berisi *state* untuk menghalangi serangan musuh sehingga kesehatan karakter tidak berkurang.

Superstate jarak jauh memiliki *superstate* yang digunakan untuk melakukan perpindahan karakter yaitu *superstate* perpindahan.

Superstate perpindahan berisi state untuk berpindah menjauh atau mendekat. Gambar dari HFSM dapat dilihat pada Gambar 3.2.



Gambar 3.2 Alur Kecerdasan Buatan

3.2.1. Superstate Jarak Jauh

Superstate jarak jauh merupakan sebuah *superstate* yang bertujuan untuk mengatur pergerakan karakter ketika jarak antara karakter >= 0.7. *Superstate* jarak jauh memiliki *superstate* yang digunakan untuk melakukan perpindahan yaitu *superstate* perpindahan. Perpindahan karakter diatur oleh *superstate* pencarian hasil *decision tree* jarak jauh.

3.2.2. Superstate Jarak Dekat

Superstate jarak dekat merupakan sebuah *superstate* yang bertujuan untuk mengatur pergerakan karakter ketika jarak antara karakter < 0.7. *Superstate* jarak dekat memiliki fungsi untuk mengurangi kesehatan musuh dan menjaga kesehatan karakter itu sendiri. Ketika karakter ingin menyerang *state* harus berada dalam *superstate* menyerang. Ketika karakter ingin bertahan *state* harus berada dalam *superstate* bertahan. Pemilihan *superstate* menyerang atau *superstate* bertahan diatur oleh *superstate* pencarian hasil *decision tree* jarak dekat.

3.2.3. Superstate Pencarian Hasil Decision Tree Jarak Jauh

Superstate pencarian hasil *decision tree* jarak jauh digunakan untuk memilih perpindahan. Superstate ini berjalan ketika karakter baru saja masuk ke dalam *superstate* jarak jauh atau ketika perpindahan yang dilakukan telah selesai. Ketika masuk ke dalam *superstate, state* memasukan nilai atribut yang akan dicari hasilnya aktif. Atribut dimasukan oleh *state* adalah atribut perpindahan musuh, atribut pemain terkena serangan, atribut kesehatan pemain dan atribut kesehatan musuh.

Setelah atribut dimasukan, hasilnya akan dicari di dalam *decision tree* jarak jauh. Hasil *output* dari *superstate* adalah mendekat atau menjauh.

3.2.4. Superstate Pencarian Hasil Decision Tree Jarak Dekat

Superstate pencarian hasil *decision tree* jarak dekat digunakan untuk memilih *superstate* menyerang atau *superstate* bertahan. Superstate ini berjalan ketika karakter baru saja masuk ke dalam *superstate* jarak dekat atau ketika aksi yang dilakukan dalam *superstate* menyerang atau *superstate* bertahan telah selesai. Ketika masuk ke dalam *superstate*, *state* memasukan nilai atribut yang akan dicari hasilnya aktif. Atribut dimasukan oleh *state* adalah atribut musuh terkena serangan, atribut pemain terkena serangan, atribut kesehatan pemain dan atribut kesehatan musuh.

Setelah atribut dimasukan hasilnya akan dicari di dalam *decision tree* jarak dekat. Hasil *output* dari *superstate* adalah menyerang atau bertahan.

3.2.5. Superstate Perpindahan

Superstate perpindahan adalah sebuah *superstate* yang menampung *state* mendekat dan menjauh. Jika *superstate* mendapat masukan mendekat, maka akan masuk ke *state* mendekat. Jika *superstate* mendapat masukan menjauh, maka akan masuk ke *state* jauh. Alur akan keluar dari *superstate* jika telah melewati 0,5 detik.

3.2.6. Superstate Bertahan

Superstate bertahan adalah sebuah *superstate* yang menampung *state* menangkis dan menjauh. Jika *superstate* mendapat masukan menjauh, maka akan masuk ke *state* menjauh. Jika mendapat masukan terkena serangan ketika dalam *state* menjauh, maka akan masuk ke *state* menangkis. Alur akan keluar dari *superstate* jika telah melewati 0,5 detik.

3.2.7. Superstate Menyerang

Superstate menyerang adalah sebuah *superstate* yang menampung *state* memilih kombo serangan, pukul dan tendang. Ketika mendapat masukan menyerang, maka akan masuk ke *state* memilih kombo serangan. Di dalam *state* kombo serangan, program memilih kombo serangan secara acak.

Kombo serangan berisi daftar serangan-serangan yang akan dilakukan. Jika serangan selanjutnya adalah pukul akan masuk ke *state* pukul. Jika serangan selanjutnya adalah tendang akan masuk ke *state* tendang. Alur akan dilakukan hingga kombo serangan habis atau kombo serangan dipatahkan.

3.3. Decision Tree

Perancangan DT dimulai dengan menyusun nilai dari setiap atribut. Setelah nilai atribut disusun nilai atribut akan dihitung berdasarkan algoritma ID3. Hasil dari perhitungan adalah sebuah *tree* yang menentukan pergerakan dari karakter.

3.3.1. Penghitungan Decision Tree Jarak Dekat

Tabel 3.1 memberikan data-data yang digunakan untuk penghitungan DT jarak dekat. Hasil dari tabel adalah karakter akan menyerang atau bertahan. Data yang dimasukan ke dalam tabel dikumpulkan dari pemain yang telah mencoba permainan.

Tabel 3.1 Nilai Atribut Decision Tree Jarak Dekat

No	Pemain Terkena Serangan	Musuh Terkena Serangan	Kesehatan Pemain	Kesehatan Musuh	Hasil
1	Ya	Tidak	Banyak	Banyak	Bertahan
2	Tidak	Ya	Banyak	Banyak	Menyerang
3	Tidak	Tidak	Sedikit	Banyak	Menyerang
4	Tidak	Tidak	Banyak	Sedikit	Menyerang

5	Ya	Ya	Banyak	Banyak	Bertahan
6	Ya	Tidak	Sedikit	Banyak	Bertahan
7	Ya	Tidak	Banyak	Sedikit	Menyerang
8	Tidak	Ya	Sedikit	Banyak	Menyerang
9	Tidak	Ya	Banyak	Sedikit	Menyerang
10	Tidak	Tidak	Banyak	Banyak	Menyerang
11	Tidak	Tidak	Sedikit	Sedikit	Menyerang
12	Ya	Ya	Sedikit	Banyak	Bertahan
13	Ya	Ya	Banyak	Sedikit	Menyerang
14	Ya	Ya	Sedikit	Sedikit	Bertahan
15	Ya	Tidak	Sedikit	Sedikit	Bertahan
16	Tidak	Ya	Sedikit	Sedikit	Menyerang

Keterangan Tabel:

- Pemain terkena serangan memiliki 2 jenis nilai yaitu ya dan tidak. Ya terjadi ketika pemain terkena serangan, sedangkan tidak terjadi ketika pemain tidak terkena serangan. Jika ya maka akan menambah kemungkinan pemain untuk mundur/menangkis sehingga tidak terkena serangan selanjutnya. Jika tidak maka akan menambah kemungkinan pemain untuk menyerang sehingga dapat memberi kerusakan lebih kepada musuh.
- Musuh terkena serangan memiliki 2 jenis nilai yaitu ya dan tidak. Ya terjadi ketika musuh terkena, serangan sedangkan tidak terjadi ketika musuh tidak terkena serangan. Jika ya maka akan menambah kemungkinan pemain untuk menyerang musuh. Jika tidak pemain akan bergantung pada atribut lain untuk pemilihan jawaban.
- Kesehatan pemain memiliki 2 jenis nilai yaitu banyak dan sedikit. Banyak terjadi ketika nilai kesehatan pemain > 50, sedangkan sedikit terjadi ketika kesehatan pemain < 51. Atribut kesehatan pemain bertujuan untuk membuat lebih berhati–hati ketika kesehatan pemain sudah sedikit.
- Kesehatan musuh memiliki 2 jenis nilai yaitu banyak dan sedikit. Banyak terjadi ketika nilai kesehatan musuh > 50, sedangkan sedikit terjadi ketika kesehatan musuh < 51. Atribut kesehatan

musuh bertujuan untuk membuat pemain lebih agresif ketika kesehatan musuh sudah sedikit.

- Hasil akhir penghitungan memiliki 2 jenis yaitu menyerang atau bertahan. Menyerang yang dimaksudkan adalah memberikan pukulan atau tendangan untuk mengurangi kesehatan musuh. Ketika bertahan pemain tidak akan terkena serangan karena otomatis menangkis jika terkena serangan.

Pada bagian ini menjelaskan hasil perhitungan *information gain* dalam betuk tabel. Alur cara penghitung akan sama seperti yang dijelaskan pada sub-bab 2.5.1.

Pada Tabel 3.2 menjelaskan hasil perhitungan dari *root*. Hasil perhitungan dikelompokan berdasarkan jenis nilai atribut.

Tabel 3.2 Hasil Perhitungan Information Gain Root dari Decision Tree Jarak Dekat

No	Nilai	Atribut	Hasil
1	Root	Pemain Terkena Serangan	0.548795(Dipilih)
2	Root	Musuh Terkena Serangan	0
3	Root	Kesehatan Pemain	0.04879498
4	Root	Kesehatan Musuh	0.04879498

Pada Tabel 3.3 Hasil Perhitungan Information Gain menjelaskan perhitungan dari *parent* pemain terkena serangan. Hasil perhitungan dikelompokan berdasarkan jenis nilai atribut.

Tabel 3.3 Hasil Perhitungan Information Gain Pemain Terkena Serangan dari Decision Tree Jarak Dekat

No	Nilai	Atribut	Hasil
1	Ya	Musuh Terkena Serangan	0
2	Ya	Kesehatan Pemain	0.3112781(Dipilih)
3	Ya	Kesehatan Musuh	0.3014441

1	Tidak	-	Menyerang(leaf)

Pada

Tabel 3.4 menjelaskan perhitungan dari *parent* kesehatan pemain. Hasil perhitungan dikelompokan berdasarkan jenis nilai atribut.

Tabel 3.4 Hasil Perhitungan Information Gain Kesehatan Pemain dari Decision Tree Jarak Dekat

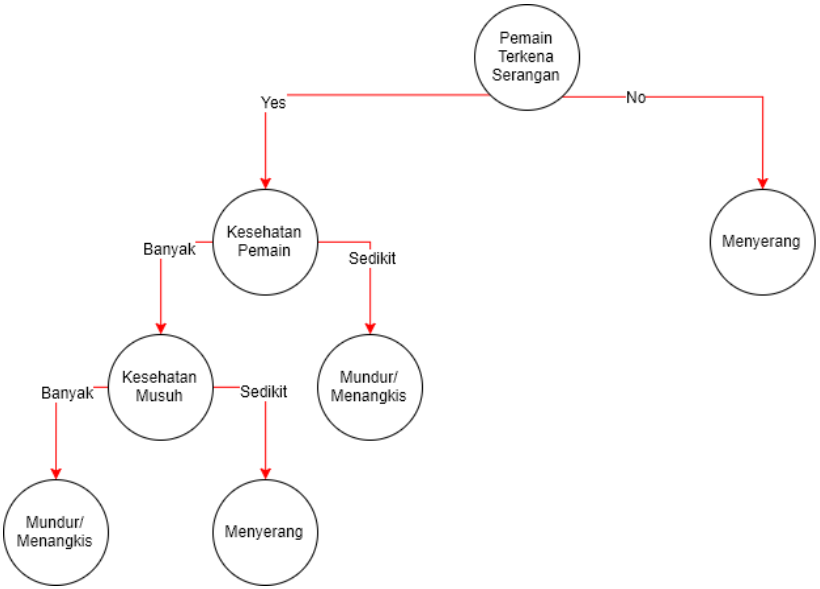
No	Nilai	Atribut	Hasil
1	Banyak	Kesehatan Musuh	1(Dipilih)
2	Banyak	Musuh Terkena Serangan	0
1	Sedikit	-	Menjauh/Menangkis(leaf)

Pada Tabel 3.5 Hasil Perhitungan Information Gain Kesehatan Musuhmenjelaskan perhitungan dari *parent* kesehatan musuh. Hasil perhitungan dikelompokan berdasarkan jenis nilai atribut.

Tabel 3.5 Hasil Perhitungan Information Gain Kesehatan Musuh dari Decision Tree Jarak Dekat

No	Nilai	Atribut	Hasil
1	Banyak	-	False(leaf)
2	Sedikit	-	True(leaf)

Bentuk *tree* dari hasil perhitungan *decision tree* jarak dekat dapat dilihat pada Gambar 3.3.



Gambar 3.3 Gambar Decision Tree Jarak Dekat

3.3.2. Penghitungan Decision Tree Jarak Jauh

Tabel 3.6 Nilai Atribut Decision Tree Jarak Jauhmemberikan data-data yang digunakan untuk penghitungan DT jarak jauh. Hasil perhitungan dari tabel akan menentukan perpindah karakter. Data yang dimasukan ke dalam tabel dikumpulkan dari pemain yang telah mencoba permainan.

Tabel 3.6 Nilai Atribut Decision Tree Jarak Jauh

No	Perpinda-han Musuh	Pemain Terkena Serangan	Kesehatan Pemain	Kesehatan Musuh	Answer
1	Menjauh	Ya	Banyak	Banyak	Mendekat
2	mendekat	Ya	Banyak	Banyak	Menjauh
3	mendekat	Ya	Sedikit	Banyak	Menjauh
4	mendekat	Ya	Banyak	Sedikit	Mendekat

5	Menjauh	Ya	Sedikit	Banyak	Menjauh
6	Menjauh	Ya	Banyak	Sedikit	Mendekat
7	mendekat	Ya	Sedikit	Sedikit	Mendekat
8	Menjauh	Ya	Sedikit	Sedikit	Menjauh
9	Menjauh	Tidak	Banyak	Banyak	Mendekat
10	mendekat	Tidak	Banyak	Banyak	Mendekat
11	mendekat	Tidak	Sedikit	Banyak	Mendekat
12	mendekat	Tidak	Banyak	Sedikit	Mendekat
13	Menjauh	Tidak	Sedikit	Banyak	Mendekat
14	Menjauh	Tidak	Banyak	Sedikit	Mendekat
15	mendekat	Tidak	Sedikit	Sedikit	Mendekat
16	Menjauh	Tidak	Sedikit	Sedikit	Mendekat

Keterangan Tabel:

- Pemain terkena serangan memiliki 2 jenis nilai yaitu ya dan tidak. Ya terjadi ketika pemain terkena serangan, sedangkan tidak terjadi ketika pemain tidak terkena serangan. Jika ya maka akan menambah kemungkinan pemain untuk mundur/menangkis sehingga tidak terkena serangan selanjutnya. Jika tidak maka akan kemungkinan besar untuk mendekat sehingga dapat memberi kerusakan lebih kepada musuh.
- Perpindahan musuh memiliki 2 jenis nilai yaitu mendekat atau menjauh. Atribut didapatkan dengan membandingkan posisi musuh sekarang dan posis musuh di iterasi sebelumnya. Ketika musuh menjauh, pemain tidak mengejar musuh karena pemain tidak dapat menyerang ketika musuh sedang menjauh.
- Kesehatan pemain memiliki 2 jenis nilai yaitu banyak dan sedikit. Banyak terjadi ketika nilai kesehatan pemain > 50 sedangkan sedikit terjadi ketika kesehatan pemain < 51. Atribut kesehatan pemain bertujuan untuk membuat lebih berhati–hati ketika kesehatan pemain sudah sedikit.
- Kesehatan musuh memiliki 2 jenis nilai yaitu banyak dan sedikit. Banyak terjadi ketika nilai kesehatan musuh > 50 sedangkan sedikit terjadi ketika kesehatan musuh < 51. Atribut kesehatan

musuh bertujuan untuk membuat pemain lebih agresif ketika kesehatan musuh sudah sedikit.

- Hasil akhir penghitungan memiliki 2 jenis yaitu mendekat dan mundur. Pergerakan mendekat dilakukan sehingga pemain dapat menyerang. Pergerakan menjauh dilakukan untuk mengulur waktu atau supaya kesehatan pemain tidak banyak berkurang.

Pada bagian ini hanya menjelaskan hasil perhitungan *information gain* dalam betuk tabel. Alur cara penghitung akan sama seperti yang dijelaskan pada sub-bab 2.5.1.

Pada Tabel 3.7 Hasil Penghitungan Informasi Gain Root dari Decision Tree Jarak Jauh menjelaskan hasil perhitungan dari *root*. Hasil perhitungan dikelompokan berdasarkan jenis nilai atribut.

Tabel 3.7 Hasil Penghitungan Informasi Gain Root dari Decision Tree Jarak Jauh

No	Nilai	Atribut	Hasil
1	Root	Perpindahan Musuh	0
2	Root	Pemain Terkena Serangan	0
3	Root	Kesehatan Pemain	0.06227887(Dipilih)
4	Root	Kesehatan Musuh	0.06227887

Pada Tabel 3.8 Hasil Perhitungan Information Gain Kesehatan Pemainmenjelaskan perhitungan dari *parent* kesehatan pemain. Hasil perhitungan dikelompokan berdasarkan jenis nilai atribut.

Tabel 3.8 Hasil Perhitungan Information Gain Kesehatan Pemain dari Decision Tree Jarak Jauh

No	Nilai	Atribut	Hasil
1	Ya	Pemain Terkena Serangan	0.1289546

2	Ya	Perpindahan Musuh	0.1254537
3	Ya	Kesehatan Musuh	0.1379254(Dipilih)
1	Tidak	Perpindahan Musuh	0.04879498(Dipilih)
2	Tidak	Pemain Terkena Serangan	0.01547543

Pada Hasil perhitungan dikelompokan berdasarkan jenis nilai atribut.

Tabel 3.9 Hasil Perhitungan Information Gain Kesehatan Musuh dari Decision Tree Jarak Jauhmenjelaskan perhitungan dari *parent* kesehatan musuh. Hasil perhitungan dikelompokan berdasarkan jenis nilai atribut.

Tabel 3.9 Hasil Perhitungan Information Gain Kesehatan Musuh dari Decision Tree Jarak Jauh

No	Nilai	Atribut	Hasil
1	Banyak	Pemain Terkena Serangan	0.3113781(Dipilih)
2	Banyak	Perpindahan Musuh	0.1241332
2	Sedikit	-	Mendekat(leaf)

Pada Tabel 3.10 Hasil Perhitungan Information Gain Perpindahan Musuh dari Decision Tree Jarak Jauhmenjelaskan perhitungan dari *parent* perpindahan musuh. Hasil perhitungan dikelompokan berdasarkan jenis nilai atribut.

Tabel 3.10 Hasil Perhitungan Information Gain Perpindahan Musuh dari Decision Tree Jarak Jauh

No	Nilai	Atribut	Hasil
1	Mendekat	Pemain Terkena Serangan	0
2	Mendekat	Kesehatan Musuh	0
1	Menjauh	Kesehatan Musuh	0.3112781

Pada Tabel 3.11 Hasil Perhitungan Information Gain Pemain Terkena Serangan dari Decision Tree Jarak Jauhmenjelaskan perhitungan dari *parent* pemain terkena serangan. Hasil perhitungan dikelompokan berdasarkan jenis nilai atribut.

Tabel 3.11 Hasil Perhitungan Information Gain Pemain Terkena Serangan dari Decision Tree Jarak Jauh

No	Nilai	Atribut	Hasil
1	Ya	Perpindahan Menjauh	1(Dipilih)
2	Tidak	-	Mendekat(leaf)

Pada Tabel 3.12 Hasil Perhitungan Information Gain Pemain Terkena Serangan(2) dari Decision Tree Jarak Jauhmenjelaskan perhitungan dari *parent* terkena serangan. Hasil perhitungan dikelompokan berdasarkan jenis nilai atribut.

Tabel 3.12 Hasil Perhitungan Information Gain Pemain Terkena Serangan(2) dari Decision Tree Jarak Jauh

No	Nilai	Atribut	Hasil
1	Ya	-	Menjauh(leaf)
2	Tidak	-	Mendekat(leaf)

Pada Tabel 3.13 Hasil Perhitungan Information Gain Kesehatan Musuh(2) dari Decision Tree Jarak Jauhmenjelaskan

perhitungan dari *parent* kesehatan musuh. Hasil perhitungan dikelompokan berdasarkan jenis nilai atribut.

Tabel 3.13 Hasil Perhitungan Information Gain Kesehatan Musuh(2) dari Decision Tree Jarak Jauh

No	Nilai	Atribut	Hasil
1	Banyak	Pemain Terkena Serangan	0(Dipilih)
2	Sedikit	-	Mendekat(leaf)

Pada Tabel 3.14 Hasil Perhitungan Information Gain Perpindahan Musuh(2) dari Decision Tree Jarak Jauhmenjelaskan perhitungan dari *parent* perpindahan musuh. Hasil perhitungan dikelompokan berdasarkan jenis nilai atribut.

Tabel 3.14 Hasil Perhitungan Information Gain Perpindahan Musuh(2) dari Decision Tree Jarak Jauh

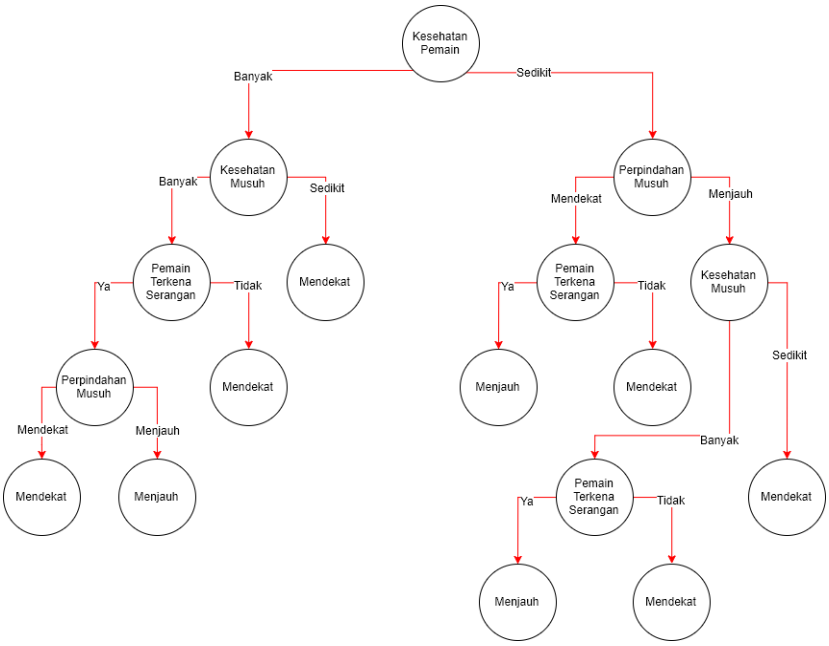
No	Nilai	Atribut	Hasil
1	Mendekat	-	Mendekat(leaf)
1	Menjauh	-	Menjauh(leaf)

Pada Tabel 3.15 Hasil Perhitungan Information Gain Pemain Terkena Serangan(3) dari Decision Tree Jarak Jauhmenjelaskan perhitungan dari *parent* pemain terkena serangan. Hasil perhitungan dikelompokan berdasarkan jenis nilai atribut.

Tabel 3.15 Hasil Perhitungan Information Gain Pemain Terkena Serangan(3) dari Decision Tree Jarak Jauh

No	Nilai	Atribut	Hasil
1	Ya	-	Menjauh(leaf)
2	Tidak	-	Mendekat(leaf)

Bentuk *tree* dari hasil penghitungan *decision tree* jarak jauh dapat dilihat pada Gambar 3.4 Gambar Decision Tree Jarak Jauh.



Gambar 3.4 Gambar Decision Tree Jarak Jauh

3.4. Perancangan Level Permainan

Pada Tabel 3.16 ini penulis akan menjelaskan perancangan *level* permainan. Setiap *level* dibedakan berdasarkan lama karakter berpikir dan kesempatan mendapat pergerakan acak. Nilai dari lama karakter berpikir dan kesempatan mendapat pergerakan acak akan dimasukan ke dalam kecerdasan buatan ketika masuk ke dalam halaman pertarungan karakter.

Tabel 3.16 Perancangan Level Permainan

No	Level	Lama Karakter Berpikir	Kesempatan Random

1	Very Hard	Close state = 0.5	0%
		Far State = 1	
2	Hard	Close state = 1	20%
		Far State = 2	
3	Normal	Close state = 1.5	40%
		Far State = 2.5	
4	Easy	Close state = 2	60%
		Far State = 3	
5	Very Easy	Close state = 3	60%
		Far State = 3.5	

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Implementasi kode program dilakukan sepenuhnya menggunakan bahasa C#.

4.1. Lingkungan implementasi

Spesifikasi perangkat keras dan perangkat lunak yang digunakan ditampilkan pada Tabel 4.1 Lingkungan implementasi Perangkat Lunak.

Tabel 4.1 Lingkungan implementasi Perangkat Lunak

Perangkat	Spesifikasi
Perangkat keras	Prosesor: Intel® Core™ i5-4200U CPU @ 1.60 GHz – 2.60 GHz Memori: 4GB
Perangkat lunak	Sistem Operasi: Windows 10 64-bit Perangkat Pengembang: Unity 5 Perangkat Pembantu: Clip Studio Paint

4.2. Implementasi Alur Kecerdasan Buatan

Pembuatan alur kecerdasan buatan merupakan inti dari tugas akhir ini. Pengerjaan alur kecerdasan buatan dimulai setelah permainan selesai dibuat. Pengerjaan alur dimulai dengan implementasi desain perancangan alur kecerdasan buatan, lalu dilanjutkan dengan implementasi *Decision Tree.*

Secara garis besar implementasi dapat dilihat pada Kode Sumber 4.1 **Implementasi** . Pada baris satu yang dilakukan adalah menghitung jarak dengan musuh dan mengkategorikan keadaan karakter. Pada baris kedua menunjukkan seberapa cepat karakter

menunggu untuk mengambil keputusan. Pada baris ketiga penghitungan pengambilan keputusan dimulai.

```
1  CekPemaindanMusuh ();
2  States ();
3  AIAgent ();
```

Kode Sumber 4.1 Implementasi Alur Kecerdasan Buatan

4.2.1. Implementasi Cek Pemain dan Musuh

Cek pemain dan musuh adalah fungsi untuk mengkategorikan jarak antara pemain dan mengkategorikan ke dalam states-states. Kode Sumber 4.2 merupakan isi kode dari cek pemain dan musuh. Pada baris pertama dapat dilihat bahwa jarak antara pemain dibandingkan dengan jarak untuk mengubah *state*, jika jarak kurang dari jarak untuk mengubah *state* maka akan masuk ke dalam keadaan jarak dekat, jika lebih akan masuk ke dalam jarak jauh. Pada baris 4 sampai 30 menyimpan keadaan musuh dan pemain untuk mencari hasil *decision tree.*

```
1  float distance = Vector3.Distance (transform.position, enState
   s.transform.position);
2  eJump = enStates.onGround;
3
4  if (states.health > 50) {
5  strPlayerHealth = "BanyakKesehatanPemain";
6  } else {
7  strPlayerHealth = "SedikitKesehatanPemain";
8  }
9
10 if (states.gettingHit) {
11 strPlayerGettingHit = "TruePemainTerkenaSerangan";
12 } else {
13 strPlayerGettingHit = "FalsePemainTerkenaSerangan";
14 }
15
```

```
16  if (enStates.health > 50) {
17  strEnemyHealth = "BanyakKesehatanMusuh";
18  } else {
19  strEnemyHealth = "SedikitKesehatanMusuh";
20  }
21
22  pastEnemtPosition = nowEnemyPosition;
23
24  nowEnemyPosition = enStates.transform.position.x;
25
26  if ((states.lookRight && pastEnemtPosition > nowEnemyPos
    ition) || (!states.lookRight && nowEnemyPosition > pastEne
    mtPosition)) {
27  enMovement = "FalseMenjauh";
28  } else {
29  enMovement = "TrueMenjauh";
30  }
31
32  if (distance < changeStateTolerance) {
33      if (aiState != AIState.resetAI) {
34          aiState = AIState.closeState;
35
36          closeCombat = true;
37      }
38  } else {
39      if (aiState != AIState.resetAI) {
40          aiState = AIState.farState;
41      }
42      if (closeCombat) {
43          if (!gotRandom) {
44              storeRandom = ReturnRandom ();
45              gotRandom = true;
46          }
47
```

```
48              if (storeRandom < 60) {
49                  Movement ();
50              }
51          }
52
53          closeCombat = false;
54      }
```

Kode Sumber 4.2 Fungsi Cek Pemain dan Musuh

4.2.2. Implementasi States Karakter

Pada Kode Sumber 4.3 memiliki kode untuk fungsi *states*. Perbedaan antara keadaan farState dan closestate adalah seberapa cepat karakter akan mengeksekusi keputusan. Lama berpikir farState dan closeState tergantung dari *level* yang dipilih seperti dalam sub-bab 3.4. Kode sumber untuk closeState dan farState dapat dilihat pada Kode Sumber 4.5 & Kode Sumber 4.4. Lama berpikir dari karakter dapat diubah diawal permainan untuk memberikan lawan bermain yang lebih mudah atau lebih susah.

```
1   switch (aiState) {
2
3       case AIState.closeState:
4           CloseState ();
5           break;
6       case AIState.farState:
7           FarState ();
8           break;
9       case AIState.resetAI:
10          ResetAI ();
11          break;
12      }
13  Block();
14  Jumping ();
```

Kode Sumber 4.3 Implementasi Fungsi States Karakter

```
1  void FarState(){
2      farTimer += Time.deltaTime;
3      if (farTimer > farRate) {
4          initiateAI = true;
5          farTimer = 0;
6      }
7  }
```

Kode Sumber 4.4 Implementasi Fungsi Far State

```
1  void CloseState(){
2      clTimer += Time.deltaTime;
3      if (clTimer > closeRate) {
4          initiateAI = true;
5          nrmTimer = 0;
6      }
7  }
```

Kode Sumber 4.5 Implementasi Fungsi Close State

Reset AI pada Kode Sumber 4.6 **Implementasi Fungsi Reset AI** bertujuan untuk mengatur keadaan karakter menjadi keadaan awal setelah karakter melakukan pemilihan pergerakan melalui kecerdasan buatan.

```
1  aiTimer += Time.deltaTime;
2
3  if (aiTimer > aiStateLife) {
4      initiateAI = false;
5      states.horizontal = 0;
6      states.vertical = 0;
7      aiTimer = 0;
8
9      gotRandom = false;
10
11     storeRandom = ReturnRandom ();
```

```
12          if (storeRandom < 50)
13                  aiState = AIState.normalState;
14          else
15                  aiState = AIState.closeState;
16
17          curNumAttacks = 1;
18          randomizeAttacks = false;
19  }
```

Kode Sumber 4.6 Implementasi Fungsi Reset AI

Fungsi *block* pada Kode Sumber 4.7 **Implementasi Fungsi Block**merupakan bertugas untuk meneruskan pergerakan perpindahan menjauh atau menangkis musuh. Fungsi ini diperlukan karena iterasi program sangat cepat sehingga sebelum keluar dari jarak pukul, karakter sudah kembali melakukan penghitungan kecerdasan buatan.

```
1   if (blocking) {
2           blockTimer = Time.deltaTime;
3           if (blockTimer < 0.4) {
4                   if (enStates.transform.position.x < transform.positi
                    on.x)
5                           states.horizontal = 1;
6                   else
7                           states.horizontal = -1;
8
9                   return;
10
11          } else {
12                  closeCombatTimer = 0;
13                  blocking = false;
14                  blockTimer = 0;
15          }
16  }
```

Kode Sumber 4.7 Implementasi Fungsi Block

Fungsi *jump* pada Kode Sumber 4.8 **Implementasi Fungsi Jump**merupakan bertugas sehingga karakter melompat dengan sendirinya secara acak setelah waktu yang di tentukan berlalu atau jika musuh melompat.

```
1  if (!eJump) {
2      float ranValue = ReturnRandom ();
3      if (ranValue < 50) {
4          jump = true;
5      }
6  }
7
8  if (jump) {
9      states.vertical = 1;
10     jRate = ReturnRandom ();
11     jump = false;
12 } else {
13     states.vertical = 0;
14 }
15
16 jtimer += Time.deltaTime;
17
18 if (jtimer > jumpRate * 10) {
19     if (jRate < 50) {
20         jump = true;
21     } else {
22         jump = false;
23 }
24
25 jtimer = 0;
26 }
```

Kode Sumber 4.8 Implementasi Fungsi Jump

4.2.3. Implementasi Menghitung Keputusan Pergerakan Karakter

Kode Sumber 4.9 merupakan isi kode untuk dari alur untuk menghitung keputusan pergerakan yang terdapat pada baris ke tiga Kode Sumber 4.1 **Implementasi** . Pada baris kesatu sistem mengecek apakah karakter akan diinisialisasi. Pada baris kedua bertujuan untuk me-*reset* karakter setelah melakukan inisialisai. Pada baris ke tiga belas dan dua puluh enam memilih menyerang atau pindah, pemilihan tersebut tergantung dari nilai *store* atau *storeRandom*. Nilai *storeRandom* didapatkan dengan menggunakan fungsi random pada baris kesembilan belas. Nilai *store* didapatkan dari fungsi *decision tree* pada baris ke tujuh atau sembilan.

Penghitungan Random untuk tingkat kesulitan dilakukan pada baris ke dua belas.

```
1  if (initiateAI) {
2      aiState = AIState.resetAI;
3
4      int newRandomVal = Random.Range (0, 5);
5
6      if (closeCombat) {
7          DecisionTreeNear ();
8      } else {
9          DecisionTreeFar ();
10     }
11
12     if (newRandomVal < (5 - randomMaxVal)) {
13         if (store < 50) {
14             Attack ();
15         } else {
16             Movement ();
17         }
18     } else {
```

```
19              storeRandom = Random.Range (0, 101);
20
21              if (storeRandom < 50) {
22                  Attack ();
23              } else {
24                  Movement ();
25              }
26          }
27  }
```

Kode Sumber 4.9 Implementasi Fungsi AI Agent

4.2.4. Implementasi Pemilihan Serangan

Kode Sumber 4.10 menjelaskan cara kerja serangan. Serangan-serangan yang akan dilakukan karakter sudah disiapkan dalam bentuk pola serangan. Pola serangan tersebut nanti akan dipilih secara acak.

```
1   if (!gotRandom) {
2       storeRandom = ReturnRandom ();
3       gotRandom = true;
4   }
5
6   if (!randomizeAttacks) {
7       numberOfAttacks = (int)Random.Range (1, 4);
8       randomizeAttacks = true;
9   }
10
11  if (curNumAttacks < numberOfAttacks) {
12      int attackNumber = Random.Range
        (0, attackPatterns.Length);
13
14      StartCoroutine (OpenAttack
        (attackPatterns [attackNumber], 0));
```

```
15
16          curNumAttacks++;
17  }
```

Kode Sumber 4.10 Implementasi Serangan

4.2.5. Implementasi Perpindahan

Kode Sumber 4.11 menjelaskan cara kerja perpindahan. Perpindahan yang dilakukan oleh karakter dipengaruhi oleh store. Jika store kurang dari 80 maka karakter akan mendekat ke musuh jika lebih karakter akan menjauh.

```
1   if (!gotRandom) {
2           storeRandom = ReturnRandom ();
3           store = storeRandom;
4   }
5
6   if (store < 80) {
7           if (enStates.transform.position.x < transform.position.x)
8                   states.horizontal = -1;
9           else
10                  states.horizontal = 1;
11  } else {
12          if (enStates.transform.position.x < transform.position.x)
13                  states.horizontal = 1;
14          else
15                  states.horizontal = -1;
16          }
17  }
```

Kode Sumber 4.11 Implemensatis Perpindahan

4.2.6. Implementasi Alur Decision Tree

Sub bab ini menjelaskan tentang implementasi *decision tree* yang dipanggil pada baris tujuh dan sembilan Kode Sumber 4.9 Implementasi . Hasil dari implementasi adalah sebuah nilai yang digunakan untuk mengubah nilai store sesuai dengan *decision tree*. Kode untuk implementasi alur dari *decision tree* jarak dekat dapat ditemukan pada Kode Sumber 4.12. Kode untuk implementasi alur dari *decision tree* jarak jauh dapat ditemukan pada Kode Sumber 4.13.

Pada bagian awal Kode Sumber 4.12 dilakukan pemilihan acak untuk melakukan tangkis atau mundur untuk mengantisipasi serangan dengan kesempatan 40%.

```
1  closeCombatTimer += Time.deltaTime;
2  if (closeCombatTimer > closeCombatRate) {
3      if (!gotRandom) {
4          storeRandom = ReturnRandom ();
5          gotRandom = true;
6      }
7
8      if (storeRandom < 40) {
9          blocking = true;
10         blockTimer = 0;
11         storeRandom = 90;
12         closeCombatTimer = 0;
13         return;
14     }
15 } else {
16     gotRandom = true;
17 }
18
19 bool attack = mytree.SearchAnswer (strPlayerGettingHit, str
   EnemyGettingHit, strPlayerHealth, strEnemyHealth);
20
21 if (attack) {
22     storeRandom = 10;
```

```
23  } else {
24      storeRandom = 90;
25      blocking = true;
26  }
```

Kode Sumber 4.12 Implementasi Alur Decision Tree Jarak Dekat

```
1  string move = mytreefar.SearchAnswer (strPlayerGettingHit,
   enMovement, strPlayerHealth, strEnemyHealth);
2
3  if (move.Equals ("Mendekat")) {
4      storeRandom = 75;
5  } else {
6      storeRandom = 90;
7  }
```

Kode Sumber 4.13 Implementasi Alur Decision Tree Jarak Jauh

4.3. Implementasi Decision Tree

Pembuatan *decision tree* merupakan salah satu inti dari tugas akhir ini. Pengerjaan *decision tree* dimulai setelah alur permainan dan alur kecerdasan buatan selesai dibuat.

4.3.1. Implementasi Tree

Sub bab ini menjelaskan tentang implementasi *tree* tempat hasil perhitungan *decision tree* akan disimpan. Pembuatan *tree* menggunakan fungsi dictionary seperti yang terlihat pada Kode Sumber 4.14 Implementasi Dictionary Nodes dan Leafs dan Kode Sumber 4.15 Implementasi Penambahan Node dan Leafs menjelaskan bagaimana cara memasukan nilai ke dalam *tree*.

```
1  public Dictionary<string,Node> nodes = new Dictionary<stri
   ng,Node> ();
2  public Dictionary<string,Leaf> leafs = new Dictionary<string
   ,Leaf> ();
```

Kode Sumber 4.14 Implementasi Dictionary Nodes dan Leafs

```
1  public void AddNode(string nodename, string value,
   string parent){
2      var nodeInfo = new Node (value, parent);
3      nodes.Add(nodename,nodeInfo);
4  }
5
6  public void AddLeaf(string value, bool finalAnswer,
   string parent){
7      var leafInfo = new Leaf (finalAnswer, parent);
8      leafs.Add(value,leafInfo);
9  }
```

Kode Sumber 4.15 Implementasi Penambahan Node dan Leafs

Pada Kode Sumber 4.16 memiliki kode untuk fungsi searchAnswer yang terdapat pada baris 19 Kode Sumber 4.12 Implementasi Alur Decision Tree dan pada baris 1 Kode Sumber 4.13. Fungsi ini digunakan untuk mencari jawaban di dalam *tree.*

```
1  public bool SearchAnswer(string Attribut1, string Attribut2, s
   tring Attribut3, string Attribut4){
2
3      string tempValue = "test";
4
5      if (lookingForRoot) {
6          int i = 0;
7          string[] nodeRoot = nodes.Where (p => p.Value.Pare
           nt == "Root").Select (p => p.Key).ToArray ();
8
9          foreach (string nodekey in nodeRoot) {
10             tempNode = nodekey;
11             i++;
12         }
13         lookingForRoot = false;
```

```
14      }
15
16      var leafvalues = leafs.Where (p => p.Value.Parent.Equals
         (tempNode)).Select (p => p.Key);
17      foreach (string value in leafvalues) {
18          if (value.Equals (Distance) || value.Equals (Block) ||
            value.Equals (GettingHit)||
            value.Equals(CurrentlyAttack)) {
19              bool endAnswer = leafs [value].FinalAnswer;
20              lookingForRoot = true;
21          return endAnswer;
22          }
23      }
24
25      var values = nodes.Where (p => p.Value.Parent.Equals(te
        mpNode)).Select (p => p.Value.Value);
26      foreach (string value in values){
27          if(value.Equals (Attribut1) || value.Equals (Attribut2)
            || value.Equals (Attribut3)|| value.Equals(Attribut4)){
28              tempValue = value;
29
30              var nodess = nodes.Where(p => p.Value.Value.
                Equals (tempValue)).Select (p => p.Key);
31              foreach (string nodekey in nodess) {
32                  tempNode = nodekey;
33                  return SearchAnswer (Distance,Block,
                    GettingHit,CurrentlyAttack);
34              }
35          }
36      }
37      return false;
38  }
```

Kode Sumber 4.16 Implementasi Mencari Jawaban di Tree

4.3.2. Implementasi Penghitungan Decision Tree

Sub bab ini menjelaskan tentang implementasi penghitungan *decision tree* yang dipanggil ketika memasuki *scenes level*. Hasil dari implementasi adalah sebuah *tree* yang akan digunakan oleh karakter.

Kode untuk implementasi hasil dari penghitungan *decision tree* dapat ditemukan pada Kode Sumber 4.17. pada baris 1 sampai 13 menjelaskan cara pemilihan *root* dari *tree*. Pada baris 15 sampai 49 menjelaskan cara pencarian *node* anak selanjutnya atau *leaf* berdasarkan nilai dari *node parent*, atribut yang telah memiliki *node* anak atau *leaf* akan ditandai dan tidak akan dipanggil lagi. Pada baris 50 sampai 59 menjelaskan iterasi dari banyaknya *node*.

1	if (!mytree.CheckIfDictioneryNotEmpty()) {
2	importantAtt = ImportantAtt (Attributes. Attribut1, "Root", Attributes.Root, importantAtt);
3	
4	importantAtt = ImportantAtt (Attributes. Attribut2, "Root", Attributes.Root, importantAtt);
5	
6	importantAtt = ImportantAtt (Attributes. Attribut3, "Root", Attributes.Root, importantAtt);
7	
8	importantAtt = ImportantAtt (Attributes. Attribut4, "Root", Attributes.Root, importantAtt);
9	
10	MakeNewNodeOnTree(importantAtt,"Root","Root");
11	importantAttIG = 0;
12	fromRoot = true;
13	}
14	

15	if (importantAtt == Attributes. Attribut1
	&& !mytree.CheckIfNodeIsParentOfNode(importantAtt.ToSt
	ring())) {

16	string value = value1;
17	Attribut1Child (importantAtt,value);
18	

19	value = value2;
20	Attribut1Child (importantAtt,value);
21	
24	}
25	
26	if (importantAtt == Attributes. Attribut2&& !mytree.CheckIfNodeIsParentOfNode(importantAtt.ToString())) {
27	string value = value1;
28	Attribut2Child (importantAtt,value);
29	
30	value = value2;
31	Attribut2Child (importantAtt,value);
32	}
33	
34	if (importantAtt == Attributes. Attribut2 && !mytree.CheckIfNodeIsParentOfNode(importantAtt.ToString())) {
35	string value = value1;
36	Attribut3Child (importantAtt,value);
37	
38	value = value2;
39	Attribut3Child (importantAtt,value);
40	}
41	
42	if (importantAtt == Attributes. Attribut2 && !mytree.CheckIfNodeIsParentOfNode(importantAtt.ToString())) {
43	string value = value1;
44	Attribut4Child (importantAtt,value);
45	
46	value = value2;
47	Attribut4Child (importantAtt,value);
48	}

```
49
50  if (childNum == 2) {
51      childNum = 0;
52      return;
53  } else {
54      if (fromRoot) {
55          for (int i = 0; i < nodeNum; i++) {
56              MakeDecision (newNode [i]);
57          }
58      }
59  }
```

Kode Sumber 4.17 Implementasi Penghitungan Decision Tree

Pada Kode Sumber 4.18 akan menjelaskan cara mencari *node* anak atau *leaf* pada *node*. Penghitungan setiap atribut dilakukan pada baris 2 sampai 7, jika atribut sudah menjadi *node* maka tidak akan dipanggil. Pada baris 11 sampai 14 merupakan pembuatan *leaf* jika nilai dari atribut sama dengan 1000. Pada baris 15 sampai 17 merupakan pembuatan *node*.

```
1   Attributes tempImportantAtt = Attributes.Attribut1;
2   if(!aldNode. Attribut2)
3       tempImportantAtt = ImportantAtt (Attributes.
        Attribut2, value, importantAtt, tempImportantAtt);
4   if(!aldNode. Attribut3)
5       tempImportantAtt = ImportantAtt (Attributes.
        Attribut3, value, importantAtt, tempImportantAtt);
6   if(!aldNode. Attribut4)
7       tempImportantAtt = ImportantAtt (Attributes.
        Attribut4 , value, importantAtt, tempImportantAtt);
8   if (tempImportantAtt == Attributes. Attribut1
    && importantAttIG != 1000) {
9       tempImportantAtt = ImportantAtt (Attributes.
        Attribut1, value, importantAtt, tempImportantAtt);
10
```

```
11  if (importantAttIG == 1000) {
12      mytree.AddLeaf (value, aldNode.final, importantAtt.ToStr
        ing());
13      childNum++;
14  } else {
15      MakeNewNodeOnTree(tempImportantAtt,value,important
        Att.ToString());
16      newNode [nodeNum] = tempImportantAtt;
17      nodeNum++;
18  }
19
20  importantAttIG = -1;
21  }
```

Kode Sumber 4.18 Implementasi Mencari Node Anak atau Leaf dari Node

Pada Kode Sumber 4.19 menjelaskan cara atribut yang terbaik dipilih dari *attribute gain* yang didapatkan pada baris satu akan dibandingkan dengan nilai atribut sebelumnya, atribut yang dipilih adalah atribut dengan nilai information gain terbesar. Jika nilai information gain adalah 1000 maka nilai atribut adalah sebuah *leaf*.

```
1   tempIG = InformationGain(attribute, value, parentAtt.ToStrin
    g());
2
3   if (tempIG < 1000) {
4       if (tempIG > importantAttIG) {
5           importantAttIG = tempIG;
6           return attribute;
7       } else {
8           return preAttribute;
9       }
10  }
11  importantAttIG = 1000;
```

```
12  return parentAtt;
```

Kode Sumber 4.19 Implementasi Pemilihan Atribut terbaik atau Leaf

Pada Kode Sumber 4.20 menjelaskan alur kalkulasi *information gain*. Pertama akan di cek apakah nilai dari *node* adalah *leaf*, jika *leaf* maka tidak akan melakukan penghitungan *information gain*. Ketika menghitung *information gain* yang pertama kali dilakukan adalah menghitung jumlah nilai atribut positif, nilai atribut negatif, total nilai atribut positif, dan total nilai atribut negatif dari atribut. Setelah itu baru dilakukan penghitung *information gain*.

```
1   calIG = CheckIfLeaf ();
2   if (calIG < 10) {
3       if (attribute == Attributes. Attribut1) {
4           IGAttribut1 ();  //menghitung jumlah attValPos,
            attValNeg, attPos, attNeg dari atribut
5           calIG = CalInformationGain (attValPos, attValNeg, 2,
            attPos, attNeg);
6       }
7       if (attribute == Attributes. Attribut2) {
8           IGAttribut2 ();  //menghitung jumlah attValPos,
            attValNeg, attPos, attNeg dari atribut
9           calIG = CalInformationGain (attValPos, attValNeg, 2,
            attPos, attNeg);
10      }
11      if (attribute == Attributes. Attribut3) {
12          IG Attribut3 ();  //menghitung jumlah attValPos,
            attValNeg, attPos, attNeg dari atribut
13          calIG = CalInformationGain (attValPos, attValNeg, 2,
            attPos, attNeg);
14      }
15      if (attribute == Attributes. Attribut4) {
```

```
16          IGAttribut4 ();  //menghitung jumlah attValPos,
            attValNeg, attPos, attNeg dari atribut
17          calIG = CalInformationGain (attValPos, attValNeg, 3,
            attPos, attNeg);
18      }
19  }
20  resetBan ();
21  return calIG;
```

Kode Sumber 4.20 Implementasi Alur Kalkulasi Information Gain dari Atribut

Sebelum menghitung jumlah atribut maupun nilai atribut, sistem perlu memberi tanda kepada nilai yang tidak akan dibaca yang dijelaskan Kode Sumber 4.21. Penandaan atribut akan diulang sampai iterasi mencapai root.

```
1   if (value == "Root") {
2       return ChooseCalIG (attribute,value);
3   }
4   if (parent.Equals(Attribut1)) {
5       for (int i = 0; i < predictionList.Count; i++) {
6           if (!value.Equals(predictionList [i].
            Attribut1.ToString())) {
7                   predictionList [i].banned = true;
8           }
9       }
10  }
11
12  if (parent.Equals(Attribut2)) {
13      for (int i = 0; i < predictionList.Count; i++) {
14          if (!value.Equals(predictionList [i].
            Attribut2.ToString())) {
15                  predictionList [i].banned = true;
16          }
```

```
17      }
18  }
19
20  if (parent.Equals(Attribut3)) {
21      for (int i = 0; i < predictionList.Count; i++) {
22          if (!value.Equals(predictionList [i].
            Attribut3.ToString())) {
23              predictionList [i].banned = true;
24          }
25      }
26  }
27
28  if (parent.Equals(Attribut4)) {
29      for (int i = 0; i < predictionList.Count; i++) {
30          if (!value.Equals(predictionList [i].
            Attribut4.ToString())) {
31              predictionList [i].banned = true;
32          }
33      }
34  }
35
36  string valParent = mytree.LookingForParent (parent);
37  if (valParent.Equals ("Root")) {
38      tempIG = ChooseCalIG (attribute, value);
39      return tempIG;
40  } else {
41      string newValue = mytree.LookingForValue (parent);
42      tempIG = InformationGain (attribute, newValue, valParent
        );
43  }
44  return tempIG;
```

Kode Sumber 4.21 Implementasi Eliminasi Nilai

Pada Kode Sumber 4.22 ini program menjalankan fungsi ChekIfLeaf di baris 1 Kode Sumber 4.20 Implementasi Alur Kalkulasi Information Gain dari Atribut. Fungsi ini untuk mengecek apakah ada leaf di atribut jika jumlah total positif atau negatif nilai dari atribut adalah leaf.

```
1  for (int i = 0; i < predictionList.Count; i++) {
2      if (!predictionList [i].banned) {
3          if (predictionList [i].attack) {
4              attPos++;
5          } else {
6              attNeg++;
7          }
8      }
9  }
10
11 if (attPos == 0 || attNeg == 0) {
12         if (attPos == 0) {
13             aldNode.final = false;
14         }
15         if (attNeg == 0){
16             aldNode.final = true;
17         }
18 }
```

Kode Sumber 4.22 Implementasi Cek Leaf

Pada Kode Sumber 4.23 menjelaskan cara penghitungan nilai atribut yang akan di hitung pada Kode Sumber 4.24 Implementasi Kalkulasi Information Gain.

```
1  for (int i = 0; i < predictionList.Count; i++) {
2      if (!predictionList [i].banned) {
3          if (predictionList [i].attribute == true) {
4              if (predictionList [i].attack) {
5                  attValPos [0]++;
```

```
6                   attPos++;
7               } else {
8                   attValNeg [0]++;
9                   attNeg++;
10              }
11          }
```

Kode Sumber 4.23 Implementasi Kalkulasi nilai Atribut

Pada Kode Sumber 4.24 menjelaskan cara kalkulasi *information gain*. Pada baris dua sistem menghitung entropi atribut. Pada baris empat entropi total positif dan negatif dikurangi entropi atribut untuk menghasilkan *information gain*.

```
1   for (int i = 0; i < attValKind; i++) {
2       entropyAtt += (((attValPos[i]+attValNeg[i])/(attPos+attNe
        g))*Entropy (attValPos[i], attValNeg[i]));
3   }
4   gain = Entropy (attPos, attNeg) - entropyAtt;
5
6   return gain;
```

Kode Sumber 4.24 Implementasi Kalkulasi Information Gain

Kode Sumber 4.25 menjelaskan cara kalkulasi entropi pada *decision tree*.

```
1   returnValue = ((-
    positive / total) * (Mathf.Log (positive / total, 2)))
     + ((-negative / total) * (Mathf.Log (negative / total, 2)));
```

Kode Sumber 4.25 Implementasi Kalkulasi Entropi

4.4. Implementasi Level Permainan

Pada sub-bab ini penulis menjelaskan implementasi dari implementasi *level***.** Pertama pemain akan mengatur nilai kecepatan berpikir dan nilai untuk mendapatkan kesempatan pergerakan acak

kecerdasan buatan. Setelah itu sistem akan memasukan nilai nilai kecepatan berpikir dan nilai untuk mendapatkan kesempatan pergerakan acak kecerdasan buatan seperti yang terlihat pada Kode Sumber 4.26.

```
1  if (charM.players [i].playerType == CharacterManager.Playe
   rBase.PlayerType.aiN) {
2      AICharacter ai = charM.players [i].playerStates.gameObj
       ect.GetComponent<AICharacter> ();
3      ai.randomMaxVal = 1; //nilai yang digunakan untuk
       mengurangi nilai acak yang memperbolehkan
       mengunakan decision tree
4      ai.mytree = mytree;
5      ai.enabled = true;
6      ai.closeRate = floatClose; //lama berpikir di closestate
7      ai.farRate = floatFar; //lama berpikir di normalstate
8
9      ai.enStates = charM.returnOppositePlater (charM.players
       [i]).playerStates;
10 }
```

Kode Sumber 4.26 Implementasi pengaturan Level Permainan

Pada Kode Sumber 4.27 menjelaskan penerapan pengaturan *level* permainan dalam pemilihan pergerakan di kecerdasan buatan. Jika nilai acak tidak termasuk maka pemilihan pergerakan akan di acak seperti pada baris 9.

```
1  int newRandomVal = Random.Range (0, 5);
2  if (newRandomVal < (5 - randomMaxVal)) {
3      if (storeRandom < 50) {
4          Attack ();
5      } else {
6          Movement ();
```

```
7       }
8   } else {
9       storeRandom = Random.Range (0, 101);
10      if (storeRandom < 50) {
11          Attack ();
12      } else {
13          Movement ();
14      }
15  }
```

Kode Sumber 4.27 Implementasi pengaturan Level Permainan diKecerdasan Buatan

[Halaman ini sengaja dikosongkan]

BAB V
PENGUJIAN DAN EVALUASI

Pada bab ini akan dijelaskan hasil uji coba dan evaluasi program yang telah selesai diimplementasi.

5.1. Lingkungan Pengujian

Lingkungan uji coba yang akan digunakan adalah,

1. Perangkat Keras
   Prosesor: Intel® Core™ i5-4200U
   CPU @ 1.60 GHz – 2.60 Ghz
   Memori: 4 GB.
2. Perangkat Lunak
   Sistem Operasi: Windows 10 Professional 64-bit
   Perangkat Pengembang: Unity3D 5

5.2. Pengujian Aturan Main Phantom Crown

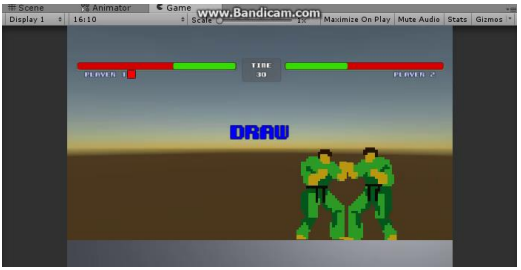
Pengujian ini dilakukan untuk mengetahui apakah aturan main yang diterapkan sudah berjalan dengan baik. Pengujian dilakukan dengan membuat sebuah skenario sebagai tolak ukur keberhasilan.

5.2.1. Skenario Uji Coba

Deskripsi	Bertujuan untuk mengetahui performa sistem dalam menerapkan aturan permainan
Langkah pengujian	1. Pemain memainkan permainan dengan tingkat kesulitan tertentu. 2. Pemain mengurangi kesehatan musuh hingga habis. 3. Sistem memunculkan tampilan pemain memenangkan permainan . 4. Pemain memainkan permainan dengan tingkat kesulitan tertentu. 5. Pemain mengurangi kesehatan musuh hingga sama dengan kesehatan pemain. 6. Sistem memunculkan tampilan seri. 7. Pemain memainkan permainan dengan tingkat kesulitan tertentu. 8. Pemain membiarkan musuh mengalahkan pemain. 9. Sistem memunculkan tampilan musuh memenangkan permainan. 10. Pemain memainkan permainan dengan tingkat kesulitan tertentu. 11. Pemain mengurangi kesehatan musuh tetapi tidak sampai habis dan membiarkan waktu bermain habis. 12. Sistem memunculkan tampilan pemain memenangkan permainan.

5.2.2. Hasil Uji Coba Aturan Main

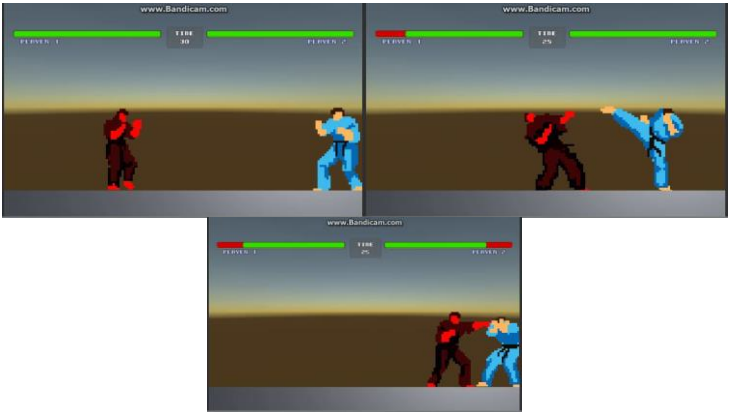
Gambar 5.1 Tampilan Dimana Pemain Menang



Gambar 5.2 Tampilan Dimana Waktu Bermain Habis dan Seri

Gambar 5.3 Tampilan Dimana Musuh Menang Tanpa Terkena Serangan

Gambar 5.4 Tampilan Dimana Pemain Menang dan Waktu Bermain Habis



Gambar 5.5 Tampilan Karakter memukul dan Menangkis Serta Waktu Bermain

Pada Gambar 5.1 Tampilan Dimana Pemain Menang memperlihatkan ketika salah karakter pemain menang dengan cara menghabiskan kesehatan musuh sebelum waktu habis. Pada Gambar 5.2 Tampilan Dimana Waktu Bermain Habis dan Seri memperlihatkan kedua kesehatan karakter sama dan waktu bermain telah habis. Pada Gambar 5.3 Tampilan Dimana Musuh Menang Tanpa Terkena Serangan memperlihatkan karakter musuh menang tanpa terkena serangan sama sekali. Gambar 5.4 Tampilan Dimana Pemain Menang dan Waktu Bermain Habis memperlihatkan ketika karakter pemain menang sebelum menghabiskan kesehatan musuh

tetapi waktu bermain telah habis. Gambar 5.5 Tampilan Karakter memukul dan Menangkis Serta Waktu Bermain menampikan karakter menyerang dan menangkis. Video uji coba aturan main dapat diakses di alamat **https://youtu.be/cXDWcbMhrwM.**

5.3. Pengujian Tingkat Kesulitan Permainan

Pengujian ini dilakukan untuk mengetahui apakah tingkat kesulitan yang diimplementasikan sudah berjalan dengan baik. Pengujian dilakukan dengan membuat sebuah skenario sebagai tolak ukur keberhasilan.

5.3.1. Skenario Uji Coba

Skenario dilakukan dengan cara menandingkan karakter dengan karakter lainya yang memiliki tingkat kesulitan berbeda, jika karakter dengan tingkat kesulitan lebih tinggi menang maka pengujian dapat dikatakan berhasil.

5.3.2. Hasil Uji Coba Tingkat Permainan

Hasil uji coba tingkat kesulitan permainan dapat dilihat di dalam Tabel 5.1.

Tabel 5.1 Hasil Uji Coba Tingkat Kesulitan Kecerdasan Buatan

No	Ronde	Tingkat Kesulitan Pemain 1	Tingkat Kesulitan Pemain 2	Pemenang	Pemenang Seharusnya
1	Ronde 1	Very Hard	Hard	Pemain 1	Pemain 1
	Ronde 2	Very Hard	Hard	Pemain 1	Pemain 1
2	Ronde 1	Very Hard	Normal	Pemain 1	Pemain 1

	Ronde 2	Very Hard	Normal	Pemain 1	Pemain 1
3	Ronde 1	Very Hard	Easy	Pemain 1	Pemain 1
	Ronde 2	Very Hard	Easy	Pemain 1	Pemain 1
4	Ronde 1	Very Hard	Very Easy	Pemain 1	Pemain 1
	Ronde 2	Very Hard	Very Easy	Pemain 1	Pemain 1
5	Ronde 1	Hard	Normal	Pemain 1	Pemain 1
	Ronde 2	Hard	Normal	Pemain 2	Pemain 1
	Ronde 3	Hard	Normal	Pemain 2	Pemain 1
6	Ronde 1	Hard	Easy	Pemain 1	Pemain 1
	Ronde 2	Hard	Easy	Pemain 1	Pemain 1
7	Ronde 1	Hard	Very Easy	Pemain 1	Pemain 1
	Ronde 2	Hard	Very Easy	Pemain 1	Pemain 1
8	Ronde 1	Normal	Easy	Pemain 1	Pemain 1
	Ronde 2	Normal	Easy	Pemain 2	Pemain 1
	Ronde 3	Normal	Easy	Pemain 1	Pemain 1
9	Ronde 1	Normal	Very Easy	Pemain 1	Pemain 1
	Ronde 2	Normal	Very Easy	Pemain 1	Pemain 1

10	Ronde 1	Easy	Very Easy	Pemain 1	Pemain 1
	Ronde 2	Easy	Very Easy	Pemain 1	Pemain 1

Berdasarkan Tabel 5.1 Hasil Uji Coba Tingkat Kesulitan karakter dengan tingkat kesulitan lebih tinggi dapat kalah dari karakter dengan tingkat kesulitan lebih rendah, jika jarak tingkat kesulitan tidak jauh misal *hard* dan *normal*. Berdasarkan 22 percobaan, terdapat 3 percobaan dimana karakter dengan tingkat kesulitan lebih rendah mengalahkan karakter dengan tingkat kesulitan lebih tinggi, sehingga persentase ketepatan tingkat kesulitan adalah 86%**.**

5.4. Pengujian Pengguna

Pengujian ini berfungsi sebagai pengujian subjektif yang bertujuan untuk mengetahui tingkat keberhasilan dalam sisi pengguna. Hal ini didapatkan dengan cara meminta penilaian dan tanggapan serta kritik dan saran dari kecerdasan buatan yang telah dibuat.

5.4.1. Skenario Uji Coba

Uji coba dilakukan dengan cara menandingkan karakter dengan tingkat kesulitan berbeda melawan manusia. Banyaknya kerusakan yang didapat musuh dan pemain akan dicatat.

Pengujian kecerdasan buatan oleh pengguna dimulai dengan memberikan informasi seputar permainan dan fitur–fitur yang dimiliki. Setelah informasi tersampaikan, pengguna akan mencoba permainan yang telah dibuat.

Jumlah pengguna yang terlibat dalam pengujian permainan berjumlah enam orang. Dalam memberikan penilaian pengguna akan diberikan formulir kuesioner penilaian. Target kepuasan kuesioner yang diharapkan mendapat nilai minimal 70%. Pada kuesioner

pengguna akan diminta kritik dan saran untuk pengembangan kecerdasan buatan untuk permainan.

5.4.2. Daftar Penguji Perangkat Lunak

Pada sub-bab ini ditunjukkan daftar pengguna yang bertindak sebagai penguji coba aplikasi yang dibangun. Daftar nama penguji aplikasi ini dapat dilihat pada Tabel 5.2 Penguji Perangkat Lunak.

Tabel 5.2 Penguji Perangkat Lunak

No	Nama	Inisial	Pekerjaan
1	Ramzy Derdianto.	RD	Mahasiswa Desain Produk ITS
2	I Gde Agung.	IGA	Mahasiswa Teknik Informatika ITS
3	Aktivano	A	Mahasiswa Sistem Kapal ITS
4	Muhamad Rizki Prawiraatmadja	MRP	Mahasiswa Teknik Informatika ITS
5	Rezky Budi Prasetyo	RBP	Mahasiswa Teknik Informatika ITS
6	Ghufron Raudya	GR	Mahasiswa Teknik Informatika ITS

5.4.3. Hasil Uji Coba Pengujian Pengguna

Hasil uji coba karakter dengan kecerdasan buatan sebagai lawan bermain untuk manusia dapat dilihat pada Tabel 5.3.

Tabel 5.3 Hasil Uji Coba Karakter Dengan Kecerdasan Buatan Sebagai Lawan Bermain

Musuh	Pemain	Kerusakan Musuh	Rata-rata Kerusakan Musuh	Kerusakan Pemain	Rata-rata Kerusakan Pemain	Pemenang
Very Hard	RD	100%	90%	60%	76,7%	Pemain
Very Hard	IGA	100%		80%		Pemain
Very Hard	A	100%		60%		Pemain
Very Hard	MRP	100%		80%		Pemain
Very Hard	RBP	60%		100%		Musuh
Very Hard	GR	80%		80%		Draw
Hard	RD	100%	97%	40%	58%	Pemain
Hard	IGA	100%		60%		Pemain
Hard	A	100%		50%		Pemain
Hard	MRP	100%		60%		Pemain
Hard	RBP	80%		60%		Pemain
Hard	GR	100%		80%		Pemain
Normal	RD	100%	100%	40%	45%	Pemain
Normal	IGA	100%		50%		Pemain
Normal	A	100%		20%		Pemain
Normal	MRP	100%		50%		Pemain
Normal	RBP	100%		60%		Pemain
Normal	GR	100%		50%		Pemain
Easy	RD	100%	100%	20%	23%	Pemain
Easy	IGA	100%		20%		Pemain
Easy	A	100%		20%		Pemain
Easy	MRP	100%		20%		Pemain

Musuh	Pemain	Kerusakan Musuh	Rata-rata Kerusakan Musuh	Kerusakan Pemain	Rata-rata Kerusakan Pemain	Pemenang
Easy	RBP	100%		40%		Pemain
Easy	GR	100%		20%		Pemain
Very Easy	RD	100%	100%	0%	10%	Pemain
Very Easy	IGA	100%		20%		Pemain
Very Easy	A	100%		0%		Pemain
Very Easy	MRP	100%		0%		Pemain
Very Easy	RBP	100%		20%		Pemain
Very Easy	GR	100%		20%		Pemain

Berdasarkan hasil uji coba pengguna melawan kecerdasan buatan dapat dilihat Tabel 5.3 dapat disimpulkan sebagai berikut:

- Musuh dengan tingkat kesulitan *easy* dan *very easy* dinilai terlalu mudah untuk pemain karena hanya dapat memberikan rata-rata kerusakan pemain dibawah 40%.
- Musuh dinilai kurang sulit untuk pemain karena hanya dapat mengalahkan pemain sekali.
- Musuh kurang dapat mempertahankan diri dari pemain karena selalu mendapat rata-rata kerusakan musuh lebih dari 80%.
- Musuh dengan tingkat kesulitan very hard dapat memberikan rata-rata kerusakan pemain cukup besar yaitu 76,7%.

Kuesioner pengguna memiliki beberapa aspek yang dipisahkan yaitu tingkat kesulitan, imersif, dan performa permainan. Sistem penilaian kuesioner didasarkan pada skala penghitungan satu sampai empat di mana skala satu menunjukkan nilai terendah dan skala empat menunjukkan skala tertinggi. Penilaian akhir kemudian dilakukan dengan menghitung berapa banyak penguji yang memilih suatu skala tertentu dan kemudian dicari nilai rata-ratanya.

Hasil kuesoner yang diberikan kepada pengguna dapat dilihat pada tabel 5.4. Cara penghitungan persentase kepuasan pengguna dari kuesioner adalah sebagai berikut:

$$H = 100\% \times \sum_{i=1}^{i=4} \frac{x_i \times i}{x \times 4} \tag{5.1}$$

Keterangan:

$\boldsymbol{x}$: Jumlah pengguna.

$\boldsymbol{x_i}$: Jumlah pengguna memilih penilaian ke-*i*.

$\boldsymbol{H}$: Merupakan rata-rata persentase penilaian.

$$J = \sum_{a=1}^{a=y} \frac{H_a}{y} \tag{5.2}$$

Keterangan:

$\boldsymbol{y}$: Jumlah soal pada parameter.

$\boldsymbol{J}$: Rata-rata total persentase penilaian parameter.

$\boldsymbol{H}$: Rata-rata persentase penilaian pada soal ke-a.

$$K = \sum_{a=1}^{a=z} \frac{J_a}{z} \tag{5.3}$$

Keterangan:

z : Jumlah parameter.

J : Rata-rata total persentase penilaian pada parameter ke-*a*.

K: Persentase keberhasilan kuesioner

Tabel 5.4 Hasil Kuesioner Pengguna

No	Parameter Tingkat Kesulitan	Penilaian				Rata-Rata Persentase Penilaian	Rata-Rata Total Persentase Penilaian Parameter
		1	2	3	4		
1	Aplikasi sudah memiliki tingkatan kesulitan yang tepat		**3**	**2**	**1**	**67%**	**66%**
2	Tingkat kesulitan very easy dapat membantu pemain memahami permainan		**3**	**3**		**63%**	
3	Tingkat kesulitan very hard dapat teman bermain yang menantang		**3**	**2**	**1**	**67%**	
	Parameter Imersif						

4	Saya merasa lebih untuk memainkan permainan phantom karena memiliki kecerdasan buatan		**4**	**2**		**58%**	**61%**
5	Saya merasakan sensasi seperti melawan manusia	**1**	**2**	**2**	**1**	**63%**	
	Parameter Performa Permainan						
6	Nilai dinamis pada permainan		**2**	**3**	**1**	**71%**	**70%**
7	Kesesuaian interaksi dengan hasil yang seharusnya		**1**	**3**	**2**	**79%**	
8	Performa atau kinerja pada permainan		**3**	**2**	**1**	**67%**	
9	Saya merasa tertarik untuk menggunakan aplikasi ini	**1**	**3**	**1**	**1**	**58%**	
10	Saya terhibur dengan aplikasi ini	**1**	**1**	**2**	**2**	**71%**	

Persentase kepuasan	**66%**

Berdasarkan Tabel 5.4 bahwa aplikasi mendapat nilai persentase 61% untuk imersif, 66% untuk tingkat kesulitan dan 70% untuk performa permainan. Sehingga dapat disimpulkan permainan belum sesuai target diatas 70% karena hanya memiliki 66% untuk nilai persentase kepuasan.

5.4.4. Kritik dan Saran Pengguna

Dalam memberikan kuesioner penilaian permainan terdapat formulir kritik dan saran untuk pengembangan permainan lebih lanjut. Kritik dan saran dapat dilihat pada Tabel 5.5.

Tabel 5.5 Kritik dan Saran

No	Nama	Kritik & Saran
1	Ramzy Derdianto.	Musuh dengan kecerdasan buatan harus dibuat lebih sulit
2	I Gde Agung.	Karakter harus memiliki lebih banyak aksi
3	Aktivano	Musuh terlalu mudah
4	Muhamad Rizki Prawiraatmadja	Spritenya dibuat lebih menarik
5	Rezky Budi Prasetyo	Lebih banyak tipe serangan
6	Ghufron Raudya	Gamenya harus dibuat lebih menarik

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan dijelaskan mengenai kesimpulan dari proses dan uji coba dari program dan saran untuk pengembangan dari program itu sendiri.

6.1. Kesimpulan

Dari hasil uji coba yang telah dilakukan, dapat diambil kesimpulan sebagai berikut :

1. Musuh dengan tingkat kesulitan *very hard* dapat memberikan rata-rata kerusakan kepada pemain manusia sebanyak 76,7%.
2. Rancangan alur kecerdasan buatan dengan HFSM dapat mengendalikan karakter dengan baik. Karakter dapat mengakses seluruh fitur karakter yang diberikan.
3. Berdasarkan hasil uji coba tingkat kesulitan, tingkat kesulitan memiliki akurasi ketepatan sebanyak 86%.
4. Berdasarkan kuesioner pengguna, aplikasi belum sesuai target diatas 70% karena hanya memiliki 66% untuk nilai persentase kepuasan. Menurut pengguna musuh terlalu mudah.

6.2. Saran

Saran yang diberikan untuk pengembangan perangkat lunak ini adalah:

1. Untuk membuat desain alur yang lebih baik, diharapkan menggunakan pemain profesional sebagai model untuk alur kecerdasan buatan.
2. Penggunaan metode pembuatan alur kecerdasan buatan yang mampu memberikan performa yang lebih baik seperti *ensemble learning*. Salah satu metode *ensemble learning* yang dapat diterapkan adalah *random forest*.
3. Untuk membuat permainan yang lebih menyenangkan, fitur karakter perlu ditambah.

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

[1] "The Next Generation 1996 Lexicon A to Z: Fighting Game". *Next Generation*. No. 15. Imagine Media. March 1996. p. 33.

[2] P. M. Kielar, O. Handel, D. H. Biedermann, and A. Borrmann, "Concurrent Hierarchical Finite State Machines for Modeling Pedestrian Behavioral Tendencies," *Transp. Res. Procedia*, vol. 2, pp. 576–584, Jan. 2014.

[3] G. Spanakis, G. Weiss, B. Boh, V. Kerkhofs, and A. Roefs, "Utilizing Longitudinal Data to Build Decision Trees for Profile Building and Predicting Eating Behavior," *Procedia Comput. Sci.*, vol. 100, pp. 782–789, 2016.

[4] Riccitiello, John (October 23, 2014). "John Riccitiello sets out to identify the engine of growth for Unity Technologies (interview)". *VentureBeat* (Interview). Interview with Dean Takahashi. Retrieved January 18, 2015.

[5] "Unity - Game Engine," *Unity*. [Online]. Available: https://unity3d.com. [Accessed: 12-Dec-2016].

[6] A. Vedaldi, B. Fukerson, K. Lenc, D. Perrone, M. Perdoch, M. Sulc dan H. Sarbotova, "vlfeat.org," VLFeat, 2007. [Online]. Available: http://www.vlfeat.org/about.html. [Diakses 06 12 2016].

[7] T. Filus, "Pengenalan Bahasa Pemrograman C#," *CodePolitan.com*. [Online]. Available: https://www.codepolitan.com/pengenalan-bahasa-pemrograman-c-587effa1cb95b. [Accessed: 08-Dec-2017].

[8] "The Gist of Hierarchical FSM | AiGameDev.com." [Online]. Available: http://aigamedev.com/open/article/hfsm-gist/. [Accessed: 21-Jan-2018].

[9] Utgoff, P. E. (1989). Incremental induction of decision trees. Machine learning, 4(2), 161-186. doi:10.1023/A:1022699900025

[10] S. J. Narayanan, R. B. Bhatt, and B. Perumal, “Improving the Accuracy of Fuzzy Decision Tree by Direct Back Propagation with Adaptive Learning Rate and Momentum Factor for User Localization,” *Procedia Comput. Sci.*, vol. 89, pp. 506–513, 2016.

BIODATA PENULIS

Dimas Rahman Oetomo atau biasa dipanggil Rahman dilahirkan di Jakarta pada tanggal 25 Agustus 1995 dan dibesarkan di Jakarta. Penulis adalah anak pertama dari emapt bersaudara.

Penulis menempuh pendidikan di SDIT PB Soedirman Jakarta (2001-2007), SMP N 20 Jakarta (2007-2010), dan SMA N 39 Jakarta (2010-2013). Setelah lulus SMA penulis melanjutkan ke jenjang perkuliahan di Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya. Bidang Studi yang diambil oleh penulis pada saat kuliah di Teknik Informatika ITS adalah Interaksi Grafika Seni.

Selama menempuh kuliah penulis aktif sebagai anggota Kendo ITS departemen luar negeri pada tahun 2014, dan ketua departemen luar negeri Kendo ITS pada tahun 2015. Penulis memiliki ketertarikan dalam dunia *video* game, olahraga dan seni lukis. Penulis dapat dihubungi melalui alamat *email* utomoe25@gmail.com.